TECHNICAL DOCUMENTATION

htf

Complete documentation.
One clear reference.

Version 4.3.1

Contents

htf - Hardware Testing Framework **11**
Test Examples 11
Getting Started 12
Example 12
Contact 13
Documentation 13
Introduction **13**
Editions **13**
Edition Comparison 14
Community Edition 14
Commercial Edition 19
ALM/QMS Integrations **19**
Matrix Requirements **19**
Installation 19
General Usage 19
Linking Implemented Tests 20
API Token 21
Usage from Command Line 21
Usage in Python 22
Python Functions 22
Options 23
Setting Extra Fields 25
Date and Time Formats 25
Proxy 26
SSL/TLS Verification 26
Debugging 27
Changelog 27
IBM® Engineering Requirements Management DOORS **27**
Installation, Activation and Upgrading **27**
Requirements 28
Installation on Windows 28
Installation on Linux 28
Installation on Raspberry Pi OS 28
Virtual environments with Virtualenvwrapper 28
Virtual environments with uv 29
Installation using Anaconda 29
Installation of htf via pip 29
Installation of htf via uv 30
Installation options 30

Installation behind a corporate proxy 31
Activation 31
Check installation 32
Upgrading 32
Installing a specific version 32
Automatic check for updates 33
Installing the community version 33

Support 33

Writing Tests 34
Test Functions 34
Test Classes 34
Legacy Tests: TestCase 35

Fixtures 47
Creating Fixtures 49
Using Fixtures 49
Fixture Decorator 49
Built-in Fixtures 49
Type Hints for fixtures 63
Example: Cache Fixture 63
Tagging Fixtures 64
Accessing Fixtures directly 65

Assertions 66
Assert Statement 66
Built-in Assertions 66

Behavior Driven Development 75
Features 76
Scenarios 76
Steps 76
Writing Features 76
Writing Scenarios 77
Example Feature 77
Complex Scenarios 77
Implementing Steps 78
Running Features 79
Steps with Parameters 79
Long Lines 80
Multi-Language Support 80
Context Fixture 81
Attach Data to Steps 81
Tagging 82
Requirements 82
Implemented Tests 83
Background Steps 83
Rules 84
Scenario Outlines 84
Scenario Outline Data Sources 85

Skipping Features, Scenarios or Scenario Outlines ... 90
Setup and Tear Down Code ... 91
Parsing Steps ... 92
Getting Available Steps and Statements ... 95
Examples ... 96

htf — Command line utility ... 96
Running tests ... 97
Specifying tests ... 97
Running multiple tests ... 98
Test-title ... 99
HTML-report ... 99
JUnit-XML-report ... 99
JSON-report ... 99
YAML-report ... 100
Report Server ... 100
Draft Reports ... 100
Minimized Report Size ... 100
Shuffle Tests ... 101
Filename-templates ... 101
Fail fast ... 101
File pattern ... 101
Tagging ... 102
Metadata ... 102
Overwriting Settings ... 102
Parameters ... 103
Bash-completion ... 103
Enable Bash-completion ... 104
Open docs ... 104
View licenses ... 104
Show version ... 105

htf.main() — Testscript utility ... 105
Running tests ... 106
Specifying tests ... 107
Test-title ... 110
HTML-report ... 110
JUnit-XML-report ... 111
JSON-report ... 111
YAML-report ... 111
Other reports ... 112
Report Server ... 113
Draft Reports ... 113
Minimized Report Size ... 113
Shuffle Tests ... 113
Filename-templates ... 113
Fail fast ... 114
File pattern ... 114
Tagging ... 114

Metadata ... 115
Parameters ... 115
htf.run() ... 115
htf.verify() ... 116
Docker Integration ... 117
Introduction ... 117
Operation Systems ... 117
Tags ... 117
Volumes ... 117
Bind Mounts ... 118
PYTHONPATH ... 118
Activation ... 118
Renewal ... 118
Interactive Mode ... 118
SSL Verification ... 118
Command Line Utility ... 119
Community Edition ... 119
Examples ... 119
Using Docker for a Project with Dependencies ... 120
Mapping Comports ... 121
Mapping USB Dongles ... 122
Dry Run ... 122
htf.dryrun() ... 122
htf dryrun ... 123
Tagging ... 123
Adding tags to testcases and tests ... 123
Logical expression to select tags ... 124
Selecting tests using tags ... 125
Showing available tags ... 125
Data Driven Testing ... 125
Code-based Examples ... 126
File-based Examples ... 128
Combinatoric Data Generation ... 132
Data Decorators ... 133
Combinatoric Decorators ... 134
File Iterators ... 135
Interactive Testing ... 136
Interaction Dialogs ... 136
Embed Images in Dialogs ... 143
Reference ... 145
Settings ... 147
JSON Example ... 148
YAML Example ... 149
Create a Settings File ... 150
htf.Settings ... 150
htf.SettingsNoSingleton ... 152

Decorators **153**
htf.test 153
htf.after 154
htf.before 154
htf.enqueue_exception 155
htf.metadata 155
htf.requirements 155
htf.implements 156
htf.meets_doors_requirements 156
htf.periodic 157
htf.raises 157
htf.skip 158
htf.skip_if 158
htf.stacktrace 160
htf.tags 160
htf.timed 160
User-Defined Decorators 161

Test Reports **161**
HTML Test Report 162
JUnit XML Test Report 166
JSON Test Report 166
YAML Test Report 166
IBM® Engineering Requirements Management DOORS Test Report 166

Signed Reports **168**
Creating signed reports 168
Checking signatures 168

Report Tool **169**
Usage 169
Convert Reports 169
Merge Reports 170
Upload Reports 170
Set the Report Title 170
Signed Reports 171
Extract Attachments 171
Extract Output 171

Community Add-ons **171**

Modbus — The Modbus Testing Framework **171**
Examples 172
Interfaces 175
Clients 184
Servers 533
Device Simulators 538
Exceptions 1014
Memory 1016
Protocol 1019
Requests and Responses 1019

Changelog 1129

HART — The HART Testing Framework 1130

HART Interface 1131
HART Frame 1133
HART Application Layer 1137
Data Types 1142
Universal Commands 1157
Common Practice Commands 1181
Common Practice Commands 1181
Device Specific Commands 1282
Protocol Factory 1287
HART-Shell 1290
HART-Monitor 1305
HART Device Communication 1308
HART Fuzzing 1311
HART Slave Simulator 1315
Changelog 1318

Communication 1319

Pluggable Notifier 1319
Notification Callbacks 1321
SSH Client 1322
SFTP Client 1322
SLIP — Serial Line Internet Protocol 1324
Changelog 1327

I/O — Input and Output 1328

Xentara — Real-Time Automation Platform 1328

Introduction 1328
Open a Connection 1328
Authentication 1328
TLS/SSL 1329
Self-signed certificates 1329
Close a Connection 1329
Reading values 1329
Writing values 1330
Subscriptions 1330
String Representation 1331
Use the Client like a Dictionary 1331
XentaraClient 1331
Element 1332
Subscription 1333
Event 1334

LabJack — DAQ devices 1334

Introduction 1334
Limitations 1334
Preparation on Linux 1334
Driver installation on Windows 1334

Supported devices ... 1335
List USB devices ... 1335
Opening USB devices ... 1335
Opening Modbus TCP devices ... 1335
Interaction with the devices ... 1336
Transactions ... 1336
Examples ... 1337
LabJack T4 ... 1340
LabJack T7/T7-Pro ... 2017
LabJack T8 ... 2769

IO — HILSTER's proprietary IO-hardware ... 3412
Introduction ... 3412
IO Mini ... 3413
IO Maxi ... 3417
IO Maxi Automation ... 3422
IO Mega ... 3428
Changelog ... 3435

Arduino — Simple IO using open source hardware ... 3437
Generic Arduino Support ... 3437
Arduino Nano ... 3437
Arduino Leonardo ... 3440
Arduino Leonardo Digital Input ... 3442

DAQmx — National Instruments DAQmx C-api wrapper ... 3443
Adapters ... 3444
Task ... 3458
Core ... 3613
Changelog ... 3613

Power Supplies ... 3614
Rohde + Schwarz HMP Series programmable power supplies ... 3614
Manson HCS-3XXX ... 3616
KORAD KA3005P ... 3619
Changelog ... 3622

Retry Functions ... 3623
Direct Calls ... 3624
Using the Decorator ... 3625
Using the Context Manager ... 3626

Issue Tracking ... 3627
JIRA (Atlassian) ... 3627

Thread — A traceable thread implementation ... 3629

Metadata ... 3631
Keywords ... 3631
Miscellaneous ... 3636

Filters — Building generic filter expressions ... 3636
NOT Filter ... 3636
AND Filter ... 3636

NAND Filter 3636
OR Filter 3637
NOR Filter 3637
XOR Filter 3637
XNOR Filter 3637
Examples 3637
Filters 3638
Changelog 3640

Terminal Colors 3640

Mixins 3721
Delegation 3721

Checksums 3723
CRC 3723
Changelog 3724

Cyber Resilience Act Conformity 3725
Statement of Conformity 3726
Security Measures 3726
Reporting Vulnerabilities 3726

Changelog 3726
htf-4.3.1 3726
htf-4.3.0 3727
htf-4.2.6 3727
htf-4.2.5 3727
htf-4.2.4 3727
htf-4.2.3 3727
htf-4.2.2 3727
htf-4.2.1 3727
htf-4.2.0 3728
htf-4.1.3 3728
htf-4.1.2 3728
htf-4.1.1 3728
htf-4.1.0 3728
htf-4.0.4 3729
htf-4.0.3 3729
htf-4.0.2 3729
htf-4.0.1 3729
htf-4.0.0 3729
htf-3.0.6 3730
htf-3.0.5 3730
htf-3.0.4 3730
htf-3.0.3 3730
htf-3.0.2 3731
htf-3.0.1 3731
htf-3.0.0 3731
htf-2.0.18 3732
htf-2.0.17 3732

htf-2.0.16 3732
htf-2.0.15 3732
htf-2.0.14 3732
htf-2.0.13 3732
htf-2.0.12 3732
htf-2.0.11 3733
htf-2.0.10 3733
htf-2.0.9 3733
htf-2.0.8 3733
htf-2.0.7 3733
htf-2.0.6 3733
htf-2.0.5 3734
htf-2.0.4 3734
htf-2.0.3 3734
htf-2.0.2 3734
htf-2.0.1 3734
htf-2.0.0 3734
htf-1.3.14 3735
htf-1.3.13 3735
htf-1.3.12 3735
htf-1.3.11 3735
htf-1.3.10 3735
htf-1.3.9 3735
htf-1.3.8 3736
htf-1.3.7 3736
htf-1.3.6 3736
htf-1.3.5 3736
htf-1.3.4 3736
htf-1.3.3 3736
htf-1.3.2 3736
htf-1.3.1 3737
htf-1.3.0 3737

htf - Hardware Testing Framework

htf is a professional **Hardware Testing Framework** based on Python to easily realize **Automated Hardware Tests**. It has a strong focus on reporting, industrial applications and functional safety and integrates seamlessly with Continuous Integration and Continuous Delivery as well as end-of-line testing systems.

What makes htf special?

- It allows you to **build teststands** using Python-based tests for **realtime testing** with diverse hardware (PLC, Development-Boards, PyVisa, Arduino etc.).
- It supports Behaviour Driven Development (BDD) to write low-code tests without prior coding experience.
- It enables you to implement powerful tests in plain Python, too.
- It provides you with the possibility to run interactive testing in case full automation is not possible and manual tests are required.
- It produces visual Test Reports which can be showed to your clients or stakeholders.
- It integrates seamlessly with QABench, which means it works smoothly with our Dashboard, allowing for **easy management and tracking** of requirements coverage.
- It can be used **for free** with the community edition.

The framework can be easily extended to your liking, either by yourself or with our support. If you need help or have questions of any kind, send us an e-mail.

If you prefer video showcases, take a look at our demo.

Test Examples

Behavior Driven Development:

```
Feature: Device Boot
  Requirements: REQ-Boot-1
  Tags: boot

  Scenario: Boot Device
    Given the device is shut down
    And the power supply is set to 0 V
    When the power supply is switched 24 V
    And the power button is set to on
    Then the device boots up
    And the device is be running after 10 seconds
```

Python:

```
@requirements("REQ-Boot-1")
@tag("boot")
async def test_boot(device, power_supply, delay):
   """
   This tests checks that the device can be booted.
   """
   with step("Shut down device"):
      await device.shut_down()
      await power_supply.power_off()

   with step("Switch on device"):
      await power_supply.power_on()
      await device.switch_on()

   with step("Check that the device boots up within 10 s"):
      delay(10)
      assert_true(device.is_up_and_running(), "The device did not boot up within 10 s")
```

Getting Started

To get started, take a look at the installation instructions, the introduction and how to write tests, fixtures and available test reports.

Learn how to run tests from the command line and Python scripts.

If you prefer a more guided introduction take a look at our HILSTER Testing Framework Demonstrator.

Example

```
import htf

def test_example(step, assertions, attachments, threads):
   with step("Tests can be structured with steps"):
      assertions.assert_true(True, "This step succeeds")

      with step("Attach a file if the outer step succeeds"):
         attachments.attach_file("graph.png", "Important graph")

   with step("Run something in the background"):
      def tick():
         while True:
            print('tick')
      threads.run_background(tick)


if __name__ == "__main__":
   htf.main()
```

Contact

The framework can be easily extended to your liking, either by yourself or with our support.

If you have any questions or ideas, please don't hesitate and e-mail us via `support@hilster.io`.

Documentation

Introduction

Welcome to the HILSTER Testing Framework documentation. HILSTER Testing Framework (htf) is a part of QABench.

HILSTER Testing Framework is a professional Python Testing Framework with a strong focus on reporting, industrial applications and functional safety. We are committed to enabling well structured and easy to understand tests. You can find a demo of the main features of HILSTER Testing Framework at htf-demo.

The HILSTER Testing Framework is developed and maintained by HILSTER. HILSTER offers continuous updates as well as customer support.

htf is free to use with a community license. Commercial licensing options are available.

Applications include: hardware-in-the-loop, hardware-software integration tests, system tests, end-of-line tests, as well as unittests and software-software integration tests.

Key features are: state-of-the-art test runner with fixtures and minimal overhead, tagging, smart reporting, interactive testing, data driven testing and behavior driven development.

htf seamlessly integrates with your existing continuous integration setup.

Working with binary data structures and communication protocols is comfortable using OSER.

For questions and suggestions, please contact us.

Have a lot of fun!

Editions

htf is offered as part of a community edition and a commercial edition.

The community edition allows private persons to use htf for free and allows up to 3 users in a commercial context.

If you need more than three users, please get a commercial edition.

Edition Comparison

	Community Edition	Commercial Edition
Use for free in private context	Yes	No
Use for free in commercial context	Yes, but only for up to 3 users	No
Number of users	Up to 3 users	Depends on the edition

Community Edition

The community edition contains a edition file for the core features of htf.

In order to use the community edition, please read about Installing the community version. Please also note our End-User License Agreement (EULA) and our Privacy Policy.

Note

End-User License Agreement (EULA) of HILSTER Testing Framework (htf)

This End-User License Agreement (“EULA”) is a legal agreement between you and HILSTER

This EULA agreement governs your acquisition and use of our HILSTER Testing Framework (htf) software (“Software”) directly from HILSTER or indirectly through a HILSTER authorized reseller or distributor (a “Reseller”).

Please read this EULA agreement carefully before completing the installation process and using the HILSTER Testing Framework (htf) software. It provides a license to use the HILSTER Testing Framework (htf) software and contains warranty information and liability disclaimers.

If you register for a free trial of the HILSTER Testing Framework (htf) software, this EULA agreement will also govern that trial. By clicking “accept” or installing and/or using the HILSTER Testing Framework (htf) software, you are confirming your acceptance of the Software and agreeing to become bound by the terms of this EULA agreement.

If you are entering into this EULA agreement on behalf of a company or other legal entity, you represent that you have the authority to bind such entity and its affiliates to these terms and conditions. If you do not have such authority or if you do not agree with the terms and conditions of this EULA agreement, do not install or use the Software, and you must not accept this EULA agreement.

This EULA agreement shall apply only to the Software supplied by HILSTER herewith regardless of whether other software is referred to or described herein. The terms also apply to any HILSTER updates, supplements, Internet-based services, and support services for the Software, unless other terms accompany those items on delivery. If so, those terms apply. This EULA was created by EULA Template for HILSTER Testing Framework (htf).

License Grant

HILSTER hereby grants you a personal, non-transferable, non-exclusive licence to use the HILSTER Testing Framework (htf) software on your devices in accordance with the terms of this EULA agreement.

You are permitted to load the HILSTER Testing Framework (htf) software (for example a PC, laptop, mobile or tablet) under your control. You are responsible for ensuring your device meets the minimum requirements of the HILSTER Testing Framework (htf) software.

You are **NOT PERMITTED** to:

- Edit, alter, modify, adapt, translate or otherwise change the whole or any part of the Software nor permit the whole or any part of the Software to be combined with or become incorporated in any other software, nor decompile, disassemble or reverse engineer the Software or attempt to do any such things
- Reproduce, copy, distribute or resell the Software
- Modify, adapt, or create derivative works of the Software without prior written consent from HILSTER

- Resell, sublicense, or redistribute the Software in any form, regardless of user or machine count
- Use the Software for any commercial purpose for more than 3 users
- Use the Software for any commercial purpose on more than 3 machines
- Allow any third party to use the Software on behalf of or for the benefit of any third party
- Use the Software in any way which breaches any applicable local, national or international law
- Use the Software for any purpose that HILSTER considers is a breach of this EULA agreement

Intellectual Property and Ownership

HILSTER shall at all times retain ownership of the Software as originally downloaded by you and all subsequent downloads of the Software by you. The Software (and the copyright, and other intellectual property rights of whatever nature in the Software, including any modifications made thereto) are and shall remain the property of HILSTER.

HILSTER reserves the right to grant licences to use the Software to third parties.

Termination

This EULA agreement is effective from the date you first use the Software and shall continue until terminated. You may terminate it at any time upon written notice to HILSTER.

It will also terminate immediately if you fail to comply with any term of this EULA agreement. Upon such termination, the licenses granted by this EULA agreement will immediately terminate and you agree to stop all access and use of the Software. The provisions that by their nature continue and survive will survive any termination of this EULA agreement.

Governing Law

This EULA agreement, and any dispute arising out of or in connection with this EULA agreement, shall be governed by and construed in accordance with the laws of de.

Note

Privacy Policy

At HILSTER License Server, accessible from https://licenses.hilster.io, one of our main priorities is the privacy of our visitors. This Privacy Policy document contains types of information that is collected and recorded by HILSTER License Server and how we use it.

If you have additional questions or require more information about our Privacy Policy, do not hesitate to contact us.

General Data Protection Regulation (GDPR)

We are a Data Controller of your information.

HILSTER legal basis for collecting and using the personal information described in this Privacy Policy depends on the Personal Information we collect and the specific context in which we collect the information:

HILSTER needs to perform a contract with you You have given HILSTER permission to do so Processing your personal information is in HILSTER legitimate interests HILSTER needs to comply with the law HILSTER will retain your personal information only for as long as is necessary for the purposes set out in this Privacy Policy. We will retain and use your information to the extent necessary to comply with our legal obligations, resolve disputes, and enforce our policies. Our Privacy Policy was generated with the help of GDPR Privacy Policy Generator and the GDPR Privacy Policy Generator.

If you are a resident of the European Economic Area (EEA), you have certain data protection rights. If you wish to be informed what Personal Information we hold about you and if you want it to be removed from our systems, please contact us.

In certain circumstances, you have the following data protection rights:

- The right to access, update or to delete the information we have on you.
- The right of rectification.
- The right to object.
- The right of restriction.
- The right to data portability
- The right to withdraw consent

Log Files

HILSTER License Server follows a standard procedure of using log files. These files log visitors when they visit websites. All hosting companies do this and a part of hosting services' analytics. The information collected by log files include internet protocol (IP) addresses, browser type, Internet Service Provider (ISP), date and time stamp, referring/exit pages, and possibly the number of clicks. These are not linked to any information that is personally identifiable. The purpose of the information is for analyzing trends, administering the site, tracking users' movement on the website, and gathering demographic information.

> **Consent**
>
> By using the community edition, you hereby consent to our Privacy Policy and agree to its terms.

Commercial Edition

To get a commercial edition, please contact us via e-mail.

ALM/QMS Integrations

HILSTER Testing Framework seamlessly integrates into many Application Lifecycle Management Systems and Quality Management Systems.

Matrix Requirements

Matrix Requirements is an Application Lifecycle Management System and Quality Management System made to design medical products.

QABench offers an integration (called `htf-matrixreq`) to enable specification of tests within Matrix Requirements, as well as sending test results back.

Installation

`htf-matrixreq` is included when `htf[full]` is installed on your system. It can also be installed explicitly with `htf[matrixreq]`.

See installation instructions for more details.

General Usage

In general, Matrix Requirements allows you to create and manage requirements. In addition, it allows you to specify tests that cover these requirements, and therefore link to them.

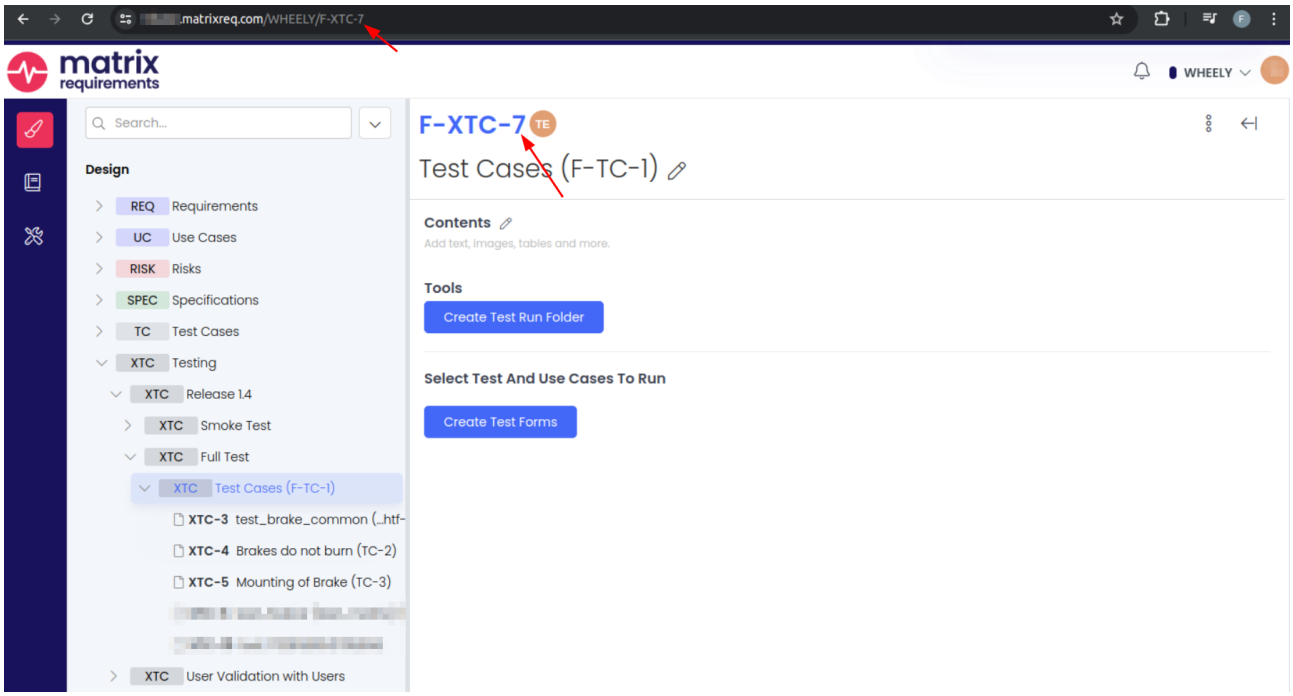
For tests, multiple test executions can be created and managed in folders.

`htf-matrixreq` is run with at least one of these folder ids, recursively collects the test execution items, finds the corresponding test specifications, runs the test and sends back the results for the implemented tests to the corresponding test execution items automatically.

`htf-matrixreq` needs at least one test execution folder in Matrix Requirements. To create that please follow the instructions in the Matrix Requirements Documentation.

Tests that will be run need to implement at least on test specification. See Linking Implemented Tests for more details.

To get the id of the test execution folder, simply click on it in the browser and get the id from the url or the title.



In the example, the test execution folder id is `F-XTC-7`.

Linking Implemented Tests

You can link the implemented Matrix Requirements test specifications in to tests in `htf` using the decorator :func:`htf.implements` for script based tests or using the `Implements:` keyword for BDD tests.

The following example shows a test with linked test specifications in Python.

```
@htf.implements("TC-1", "https://<user>.matrixreq.com/<project>/TC-2")
def test_testcase_1_and_2(step):
    with step("First test step", expected_result="This step succeeds"):
        assert True

    with step("Second test step", expected_result="This step succeeds, too"):
        assert True
```

The following example shows a corresponding test in BDD.

```
Feature: Feature with implemented tests

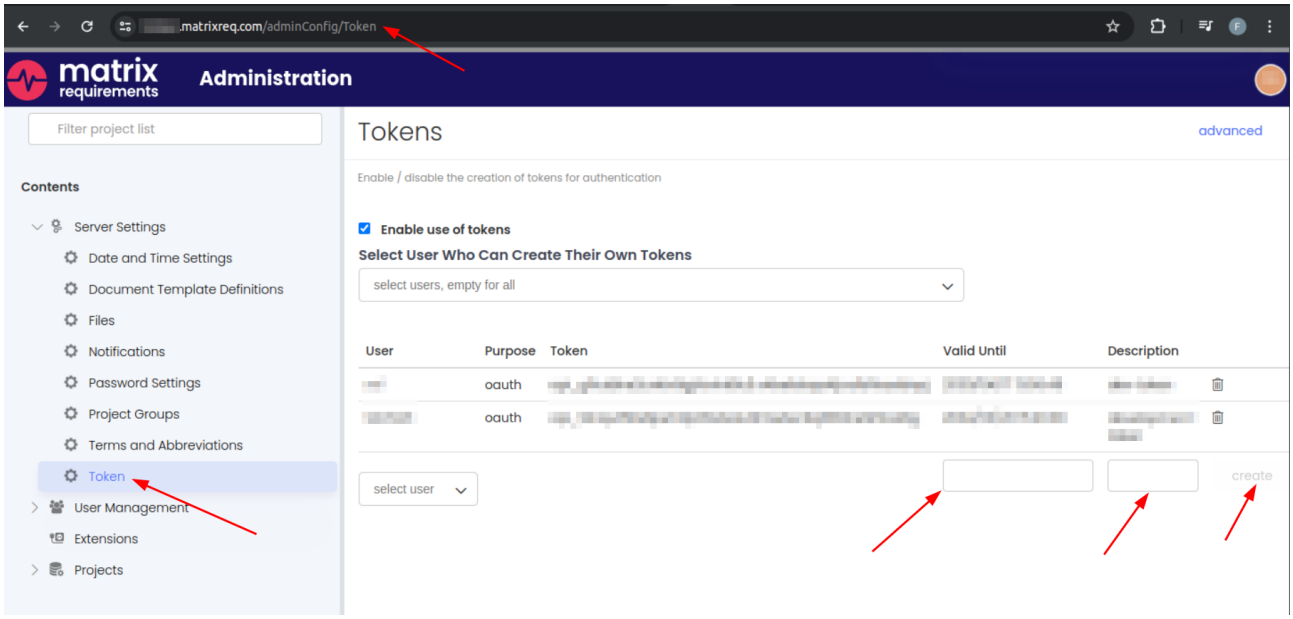
    Implements: TC-1, https://<user>.matrixreq.com/<project>/TC-2

    Scenario: Scenario with implemented tests
        Implements: TC-3
        # ...
```

API Token

`htf-matrixreq` uses the Rest API of Matrix Requirements and thus needs an API token to function. To create an API token, login into Matrix Requirements and navigate to `https://<user>.matrixreq.com/adminConfig/Token`.

Create an API token and store it.



Usage from Command Line

`htf-matrixreq` supports the same command line arguments as `htf` does and adds a few other.

For example to run all tests in the folder `tests` for the test execution folder ids `F-XTC-1` and `F-XTC-7`, run:

```
htf-matrixreq --items F-XTC-1,F-XTC-7 tests/
```

A full example in which all options are set via command line would be:

```
htf-matrixreq \
    --host https://<user>.matrixreq.com \
    --project <project> --items F-XTC-1,F-XTC-7 \
    --token <token> \
    --set Tester=tester \
    --set Version=1.2.3 \
    tests/
```

This example also sets the extra fields `Version` and *Tester* ` within the Matrix Requirements test execution item, and gets all other options from command line.

Usage in Python

`htf.matrixreq.main` supports the same parameters as `htf.main` does and adds a few other.

```
from htf.matrixreq import main

if __name__ == "__main__":
    main(
        host="https://<user>.matrixreq.com",
        project="<project>",
        token="<token>",
        items=["F-XTC-1", "XTC-7"],
        extra_fields={
            "Tester": "tester",
            "Version": "1.2.3",
        },
        tests=["tests/"],
    )
```

`htf.matrixreq.run` is an alias for `htf.matrixreq.main`.

Python Functions

```
htf.matrixreq.main(host: str | None = None, project: str | None = None, items: List[str] |
None = None, token: str | None = None, extra_fields: Dict[str, str] | None = None, insecure:
bool = False, date_format: str = 'yyyy/MM/dd', date_time_format: str = 'yyyy/MM/dd HH:mm:ss',
*htf_args: Any, **htf_kwargs: Any) → int
```

The htf.matrix.main() and `htf-matrixreq` command line utility is used to run tests and send the results back to Matrix ALM/QMS.

Parameters:

- **project** – the name of the project
- **items** – the items to search for test execution item
- **token** – the REST API token
- **extra_fields** – extra fields to set for updated test execution items
- **insecure** – if set to `True` SSL/TLS certificates are not checked
- **(default** (*date_time_format*) – “yyyy/MM/dd”): the date format for the project
- **(default** – “yyyy/MM/dd HH:mm:ss”): the date time format for the project
- ***htf_args** – htf positional arguments
- ****htf_kwargs** – htf keyword arguments

Returns:
the return code of `htf.main()` plus the number of errors and warnings produced by this tool.
Return type:
int

```
htf.matrixreq.run(host: str | None = None, project: str | None = None, items: List[str] |
None = None, token: str | None = None, extra_fields: Dict[str, str] | None = None, insecure:
bool = False, date_format: str = 'yyyy/MM/dd', date_time_format: str = 'yyyy/MM/dd HH:mm:ss',
*htf_args: Any, **htf_kwargs: Any) → int
```

The htf.matrix.main() and `htf-matrixreq` command line utility is used to run tests and send the results back to Matrix ALM/QMS.

Parameters:

- **project** – the name of the project
- **items** – the items to search for test execution item
- **token** – the REST API token
- **extra_fields** – extra fields to set for updated test execution items
- **insecure** – if set to `True` SSL/TLS certificates are not checked
- **(default** (*date_time_format*) – "yyyy/MM/dd"): the date format for the project
- **(default** – "yyyy/MM/dd HH:mm:ss"): the date time format for the project
- ***htf_args** – htf positional arguments
- ****htf_kwargs** – htf keyword arguments

Returns:
the return code of `htf.main()` plus the number of errors and warnings produced by this tool.
Return type:
int

Options

To use `htf-matrixreq`, you need to specify the hostname, the project name, the API token and, if needed, the insecure flag.

These options can either be set via command line options, by storing them in `htf.settings`, and by setting them via environment variables.

Environment variables are overwritten by `htf.settings`, whereas command line options overwrite everything else.

Command Line Options

To specify the hostname via command line, use `--host <host>`.

To specify the project name via command line, use `--project <project>`.

To specify the API token via command line, use `--token <API Token>`.

The full list of command line options is:

```
usage: htf-matrixreq [-h] [--host HOST] [--project PROJECT] --items ITEMS [--token TOKEN] [--
insecure]
                     [--set field1=value1,field2=value2,...,fieldN=valueN] [--date-format
DATE_FORMAT]
```

```
                    [--date-time-format DATE_TIME_FORMAT]

htf-matrixreq - HILSTER Testing Framework integration for Matrix Requiements ALM/QMS System
(HILSTER, https://hilster.io)

options:
  -h, --help            show this help message and exit
  --host HOST           The host of the Matrix Requirements project (required)
  --project PROJECT     The name of the Matrix Requirements project (required)
  --items ITEMS         The items to recursively search for tests in. Can be a comma-separated list
of items, e.g.
                        --item 'F-XTC-1,F-XTC-2'. (required, can be repeated)
  --token TOKEN         The REST API token to use (required)
  --insecure            Disable SSL/TLS certificate check (can be useful behind some proxy servers)
  --set field1=value1,field2=value2,...,fieldN=valueN
                        Set extra fields on every item, e.g. Tester, Version, etc. Can be repeated.
  --date-format DATE_FORMAT
                        The date format for the project.
  --date-time-format DATE_TIME_FORMAT
                        The date time format for the project.
```

Settings

You can specify hostname, project name, API token and the insecure flag in `htf.settings`, too.

To do that add a section in the settings called `matrixreq`.

YAML Example:

```yaml
matrixreq:
    host: https://<user>.matrixreq.com
    project: <project>
    token: <token>
    date_format: "yyyy/MM/dd"
    date_time_format: "yyyy/MM/dd HH:mm:ss"
    extra_fields:
        Tester: <tester>
        Version: <version>
    insecure: false  # true|false
```

JSON Example:

```json
"matrixreq": {
    "host": "https://<user>.matrixreq.com",
    "project": "<project>",
    "token": "<token>",
    "date_format": "yyyy/MM/dd",
    "date_time_format": "yyyy/MM/dd HH:mm:ss",
    "extra_fields" {
        "Tester": "<tester>",
        "Version": "<version>"
    },
```

```
    "insecure": false
}
```

You can leave out any value if it is not needed, or if it is specified elsewhere.

Environment Variables

The following tables shows the supported environment variables and their meaning.

Environment Variable	Meaning
MATRIXREQ_HOST	Hostname, e.g. https://<user>.matrixreq.com
MATRIXREQ_INSECURE	Disable SSL/TLS certificate verification. Any value is supported.
MATRIXREQ_PROJECT	Project Name
MATRIXREQ_TOKEN	API Token, e.g. <token>
MATRIXREQ_DATE_FORMAT	The date format. Default: “yyyy/MM/dd”
MATRIXREQ_DATE_TIME_FORMAT	The date time format. Default: “yyyy/MM/dd HH:mm:ss”

Setting Extra Fields

You can set extra fields within Matrix Requirements test execution items such as `Tester` or `Version` when sending test results back to test execution items.

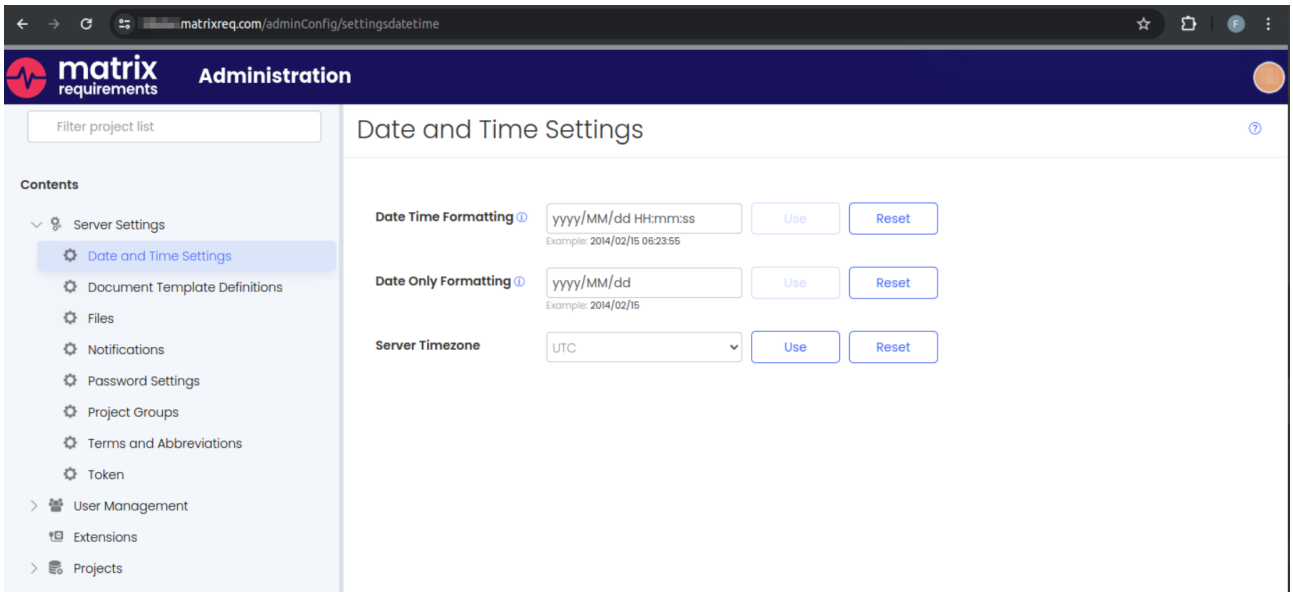
To do this in the command line, add `--set name=value` pairs. For example in order to set the `Version` to `1.2.3`, use `--item Version=1.2.3`.

You can also set these extra values using `htf.settings`.

Date and Time Formats

`htf-matrixreq` needs to set the test execution timestamps in the specified format set within the Matrix Requirements project.

To view and change the formats, login into Matrix Requirements and navigate to `https://<user>.matrixreq.com/adminConfig/settingsdatetime`.



If the formats differ the test execution timestamps will not be set.

Proxy

You can specify proxies to use by setting the environment variable `https_proxy` to `https://<username>:<password>@<host>:<port>`.

On Linux enter:

```
export https_proxy=https://<username>:<password>@<host>:<port>
```

and on Windows enter:

```
set https_proxy=https://<username>:<password>@<host>:<port>
```

SSL/TLS Verification

In some circumstances it might be necessary to disable SSL/TLS verification. To do this simply add the option `--insecure` to `htf-matrixreq` or set the environment variable `MATRIXREQ_INSECURE` to any value.

On Linux enter:

```
export MATRIXREQ_INSECURE=true
```

and on Windows enter:

```
set MATRIXREQ_INSECURE=true
```

Debugging

To enable debugging of HTTPS communication set the environment variable `MATRIXREQ_DEBUG` to any value and `htf-matrixreq`.

On Linux enter:

```
export MATRIXREQ_DEBUG=true
```

and on Windows enter:

```
set MATRIXREQ_DEBUG=true
```

Changelog

htf-matrixreq-1.1.0

- ship `cp312-abi3` wheels that are compatible with Python 3.12 and newer versions

htf-matrixreq-1.0.3

- implement Cyber Resilience Act Conformity (CRA)
- update dependencies

htf-matrixreq-1.0.2

- add support for Python 3.13

htf-matrixreq-1.0.1

- fix setting of actual results

htf-matrixreq-1.0.0

- initial integration

IBM® Engineering Requirements Management DOORS

HILSTER Testing Framework offers an integration to enable specification of tests and requirements in IBM® Engineering Requirements Management DOORS as well as sending test results back.

See IBM® Engineering Requirements Management DOORS Test Report for more details.

Installation, Activation and Upgrading

Requirements

To install `htf`, Python 3.12 or newer is required.

`htf` is tested on Microsoft Windows and all Linux versions that support manylinux wheels. It also runs on `Raspberry Pi` computers.

Wheels for Python 3 on Linux are shipped using recent `manylinux` and `abi3` formats for `Python 3.12` and newer for 64-bit machines only.

We host our own pypi server.

For further information regarding the support of your specific project requirements, please contact our Support.

Installation on Windows

If Python is not already installed, please download and install Python using the official CPython installers or use a CPython distribution of your choice, e.g. WinPython.

It is recommended to install Python in the typical paths, e.g. `C:\Python312\` for Python 3.12 respectively.

If you want to install `htf` via `pip`, you need to add the `Scripts` folder of the Python distribution to your `%PATH%` environment variable (there is an option in the installer). `pip` is shipped with recent Python Windows distributions.

Installation on Linux

On most Linux distributions, at least one version of Python should be installed by default. Since 2020, this version should be Python 3.

`pip` can be installed by installing the packages `python3-pip` for Python 3.12, respectively. There are no other dependencies that need to be installed on Linux.

On Ubuntu for example, all dependencies should be satisfied using:

```
sudo apt install python3-pip
```

Installation on Raspberry Pi OS

You can easily install HILSTER Testing Framework on a Raspberry Pi OS using the 64 bit installation image available at downloads page.

Virtual environments with Virtualenvwrapper

It is generally recommended to separate project dependencies into different virtual environments.

More information about setting up and using virtual environments on Linux can be found in the python-guide and for Windows in the virtualenv docs.

It is recommended to use the `virtualenvwrapper` package, which provides functions that allow for a convenient handling of virtual environments. Further information can be found in the virtualenwrapper docs.

There is also a Windows port called virtualenvwrapper-win.

Virtual environments with uv

uv supports virtual environments, too.

To create a virtual environment, run:

```
uv venv <virtual-environment-name>
```

To activate the virtual environment, follow the activation instructions.

Installation using Anaconda

Alternatively Anaconda can be used as a multi-platform Python-Platform. Installers for Windows and Linux are provided by Anaconda. It is pre-packaged with multiple libraries especially from the data science community. Before moving on please download and install Anaconda on your system.

After the base install it is again recommended to separate project dependencies into different virtual environments. You can create a `conda` environment using the Anaconda Prompt:

```
conda create --name environment_name python
conda activate environment_name
```

Further references to the Anaconda Prompt can be found in the Anaconda docs.

You can then proceed to install `htf` via `pip`.

Installation of htf via pip

Given that Python is properly set up and the virtual environment of your choice is activated, you can simply install `htf` using pip:

```
pip install --index-url https://hilster.io 'htf[full]'
```

To install the community version, run:

```
pip install --index-url https://hilster.io htf-community
```

Herein, `--index-url https://hilster.io` adds the HILSTER Python package repository to the search index of pip so that the HILSTER packages can be discovered.

Depending on your installation, you might have to replace `pip` with `pip3`.

Installation of htf via uv

`htf` can also be installed using uv.

To install `htf`, run:

```
uv pip install --index-url https://hilster.io 'htf[full]'
```

To install the community version, run:

```
uv pip install --index-url https://hilster.io htf-community
```

Herein, `--index-url https://hilster.io` adds the `HILSTER Python package repository` to the search index so that the HILSTER packages can be discovered.

Installation options

`htf` comes with submodules that can be installed manually or using installation options.

Installation options are put behind the package in the `pip` prompt.

For example

```
pip install -i https://pypi.hilster.io htf
```

only installs the core of `htf`.

```
pip install -i https://pypi.hilster.io htf[full]
```

installs all possible options.

There is an installation option for all submodules, e.g. the option for `htf-modbus` is `[modbus]` and for `htf-hart` you can use `[hart]`.

Installation options can be combined as a coma-separated list:

```
pip install -i https://pypi.hilster.io htf[modbus,hart]
```

installs `htf-modbus` and `htf-hart`.

Another option is `htf[oser]` to install `oser` directly.

Installation behind a corporate proxy

If you are behind a corporate proxy, there are several ways to tell `pip` about the proxy url and your credentials.

The simplest solution is to just use the `--proxy` argument

```
pip install -i https://pypi.hilster.io --proxy=http://<username>:<password>@<host>:<port> htf
```

If you want to persist the `proxy` settings, it is recommended to store them in the pip configuration file `pip.ini`, which is stored in the `pip` folder in your home directory.

```
[global]
proxy = http://<username>:<password>@<host>:<port>
```

Another solution is use the environment variable `https_proxy`.

On Linux enter:

```
export https_proxy=https://<username>:<password>@<host>:<port>
```

and on Windows enter:

```
set https_proxy=https://<username>:<password>@<host>:<port>
```

and continue with the installation without the `--proxy` option.

Activation

HILSTER's license server is used for remote activation.

Activation is not needed if you use the community edition, since a license is already included.

To activate non-community versions of the software, you need one or more activation keys that HILSTER sent you.

An activation key consists of `16` upper case letters in `4` groups concatenated by dashes.

```
hlm activate ABCD-EFGH-IJKL-MNOP
```

You can also activate more then one activation key at a time.

```
hlm activate ABCD-EFGH-IJKL-MNOP QRST-UVWX-YZAB-CDEF  # ...
```

Activation creates a license that is stored locally.

To renew the licenses once activated, simply run

```
hlm renew
```

When you are behind a proxy, you have to set the environment variable `https_proxy` like in the installation process.

Every virtual environment has to be activated separately.

Check installation

To list your licenses, run

```
hlm list
```

If htf prints a version, it is unlockable

```
htf version  # prints the version
```

Upgrading

To upgrade add the `-U` option:

```
pip install -i https://pypi.hilster.io -U htf
```

If you are using a proxy you must add the `--proxy` option or set environment variables like described above.

Installing a specific version

To install a specific version use

```
pip install -i https://pypi.hilster.io htf==2.0.0
```

for example.

If you are using a proxy you must add the `--proxy` option or set environment variables like described above.

Automatic check for updates

`htf` checks for available updates at the end of a test run automatically. This can be disabled by setting the `HTF_DISABLE_UPDATE_CHECK` environment variable.

On Linux enter:

```
export HTF_DISABLE_UPDATE_CHECK=1
```

and on Windows enter:

```
set HTF_DISABLE_UPDATE_CHECK=1
```

Installing the community version

To install the community version, run

```
pip install -i https://pypi.hilster.io htf-community
```

Manual activation is not needed because the community license is already included and the activation will be performed automatically.

Make sure to use a recent Python version to install the latest version for `htf-community`. Read installation requirements for more information.

The available wheels and supported Python versions can also be found on HILSTER's PyPI server, too.

Support

Don't panic — we can help you!

Send us an e-mail.

Alternatively, contact our support team by running

```
support
```

from the command line and following the instructions.

Writing Tests

htf includes a state of the art test runner that supports three different types of tests: functions, coroutines and methods in simple classes or subclasses of `htf.TestCase`.

Each function or method or coroutine that starts with `test` or that is decorated with `htf.test` is assumed to be a test and is collected.

We recommend to use functions, coroutines or simple classes to build tests.

All tests can use Fixtures to set up their test environment.

Tests use Assertions to assert conditions of your tests.

Test Functions

The simplest way to create a test is to define a function. The function is assumed to be a test if the name starts with `test`, or the function is decorated with `htf.test`.

Example:

```
def test_function():
    pass

async def test_async_function():
    pass

@htf.test
def function():
    pass

@htf.test
def async_function():
    pass

if __name__ == "__main__":
    htf.main()
```

Test Classes

A class is a container for multiple test methods. The method is assumed to be a test if the name starts with `test`, or the method is decorated with `htf.test`.

Example:

```
class Tests:
    def test_method(self):
        pass

    async def test_async_method(self):
        pass

    @htf.test
    def method(self):
        pass

    @htf.test
    async def async_method(self):
        pass
```

Legacy Tests: TestCase

Unittest-like tests are also supported by subclassing `htf.TestCase`.

```
class LegacyTests(htf.TestCase):
    def test_method(self):
        pass

    async def test_async_method(self):
        pass

    @htf.test
    def method(self):
        pass

    @htf.test
    async def async_method(self):
        pass
```

`htf.TestCase.setUp`, `htf.TestCase.tearDown`, `htf.TestCase.setUpClass` and `htf.TestCase.tearDownClass` are also supported. These tests cannot be run with unittest as a test runner.

```
class htf.TestCase(method_name: str = 'run_test')
```

`TestCase` is the basic test case class for the HILSTER Testing Framework that should be used with inheritance.

Parameters:
method_name='run_test' – the name of the test method to be run by the test runner.

```
static assert_almost_equal(a: float, b: float, places: int | None = None, delta: float |
None = None, message: str | None = None) → None
```

Assert that `a` and `b` are almost equal. Useful to compare floating point numbers. Comparison is realized either using a maximum `delta` or the number of decimal `places`

Parameters:

- **a** – number
- **b** – number
- **places=None** – the number of decimal places for comparison. If `None` `7` decimal places are used.
- **delta=None** – the maximum delta
- **message=None** – the message to prepend to the error message

Raises:
AssertionError – if `a` and `b` are not almost equal.

```
static assert_dict_equal(a: Dict, b: Dict, message: str | None = None) → None
```

Assert that `a` equals `b` and that both are of type `dict`.

Parameters:

- **a** – a dict
- **b** – a dict
- **message=None** – the message to prepend to the error message

Raises:
AssertionError – if `a` and `b` are not equal.

```
static assert_equal(a: Any, b: Any, message: str | None = None) → None
```

Assert that `a` equals `b`. Tries to find a registered equality asserter and uses the base assert if no special assert is registered.

Parameters:

- **a** – an object
- **b** – an object
- **message=None** – the message to prepend to the error message

Raises:
AssertionError – if `a` and `b` are not equal.

```
static assert_false(expression: bool, message: str | None = None) → None
```

Assert that `expression` is `False`.

Parameters:

- **expression** – the boolean expression to be validated
- **message=None** – the message to prepend to the error message

Raises:
AssertionError – if `expression` is not `False`

```
static assert_greater(a: Any, b: Any, message: str | None = None) → None
```

Assert that `a` is greater than `b`.

Parameters:

- **a** – a comparable object
- **b** – a comparable object
- **message=None** – the message to prepend to the error message

Raises:
AssertionError – if `a` is not greater than `b`

```
static assert_greater_equal(a: Any, b: Any, message: str | None = None) → None
```

Assert that `a` is greater or equal to `b`.

Parameters:

- **a** – a comparable object
- **b** – a comparable object
- **message=None** – the message to prepend to the error message

Raises:
AssertionError – if `a` is not greater or equal to `b`

```
static assert_in(member: Any, container: Iterable, message: str | None = None) → None
```

Assert that `container` contains `member`.

Parameters:

- **member** – the object that shall be included in `container`
- **container** – an iterable to search for `member`
- **message=None** – the message to prepend to the error message

Raises:
AssertionError – if `member` is not included in `container`

```
static assert_is(a: Any, b: Any, message: str | None = None) → None
```

Assert that `id(a)` equals `id(b)`.

Parameters:

- **a** – an object
- **b** – another object
- **message=None** – the message to prepend to the error message

Raises:
AssertionError – if `id(a)` does not equal `id(b)`

```
static assert_is_instance(obj: Any, cls: Type, message: str | None = None) → None
```

Assert that `obj` is an instance of `cls`.

Parameters:

- **obj** – an instance
- **cls** – a class `obj` shall be an instance of
- **message=None** – the message to prepend to the error message

Raises:
AssertionError – if `obj` is not an instance of `cls`.

```
static assert_is_none(obj: Any, message: str | None = None) → None
```

Assert that `obj` is `None`.

Parameters:

- **obj** – an object
- **message=None** – the message to prepend to the error message

Raises:
AssertionError – if `obj` is not `None`

```
static assert_is_not(a: Any, b: Any, message: str | None = None) → None
```

Assert that `id(a)` does not equal `id(b)`.

Parameters:

- **a** – an object
- **b** – another object
- **message=None** – the message to prepend to the error message

Raises:
AssertionError – if `id(a)` equals `id(b)`

```
static assert_is_not_instance(obj: Any, cls: Type, message: str | None = None) → None
```

Assert that `obj` is an instance of `cls`.

Parameters:

- **obj** – an instance
- **cls** – a class `obj` shall be an instance of
- **message=None** – the message to prepend to the error message

Raises:
AssertionError – if `obj` is not an instance of `cls`.

```
static assert_is_not_none(obj: Any, message: str | None = None) → None
```

Assert that `obj` is not `None`.

Parameters:

- **obj** – an object
- **message=None** – the message to prepend to the error message

Raises:
AssertionError – if `obj` is `None`

```
static assert_less(a: Any, b: Any, message: str | None = None) → None
```

Assert that `a` is less than `b`.

Parameters:

- **a** – a comparable object
- **b** – a comparable object
- **message=None** – the message to prepend to the error message

Raises:
AssertionError – if `a` is not less than `b`

```
static assert_less_equal(a: Any, b: Any, message: str | None = None) → None
```

Assert that `a` is less or equal to `b`.

Parameters:

- **a** – a comparable object
- **b** – a comparable object
- **message=None** – the message to prepend to the error message

Raises:
AssertionError – if `a` is not less or equal to `b`

```
static assert_list_equal(a: List, b: List, message: str | None = None) → None
```

Assert that `a` equals `b` and that both are of type `list`.

Parameters:

- **a** – a list
- **b** – a list
- **message=None** – the message to prepend to the error message

Raises:
AssertionError – if `a` and `b` are not equal.

```
static assert_multi_line_equal(a: str, b: str, message: str | None = None) → None
```

Assert that `a` equals `b` and that both are of type `str`.

Parameters:

- **a** – a string
- **b** – a string
- **message=None** – the message to prepend to the error message

Raises:
AssertionError – if `a` and `b` are not equal.

```
static assert_not_almost_equal(a: float, b: float, places: int | None = None, delta:
float | None = None, message: str | None = None) → None
```

Assert that `a` and `b` are not almost equal. Useful to compare floating point numbers. Comparison is realized either using a maximum `delta` or the number of decimal `places`

Parameters:

- **a** – number
- **b** – number
- **places=None** – the number of decimal places for comparison. If `None` `7` decimal places are used.
- **delta=None** – the maximum delta
- **message=None** – the message to prepend to the error message

Raises:
AssertionError – if `a` and `b` are almost equal.

```
static assert_not_equal(a: Any, b: Any, message: str | None = None) → None
```

Assert that `a` does not equal `b`.

Parameters:

- **a** – an object
- **b** – an object
- **message=None** – the message to prepend to the error message

Raises:
AssertionError – if `a` and `b` are equal.

```
static assert_not_in(member: Any, container: Iterable, message: str | None = None) →
None
```

Assert that `container` does not contain `member`.

Parameters:

- **member** – the object that shall not be included in `container`

- **container** – an iterable to search for `member`
- **message=None** – the message to prepend to the error message

Raises:
AssertionError – if `member` is included in `container`

```
static assert_not_is_instance(obj: Any, cls: Type, message: str | None = None) → None
```

Assert that `obj` is an instance of `cls`.

Parameters:

- **obj** – an instance
- **cls** – a class `obj` shall be an instance of
- **message=None** – the message to prepend to the error message

Raises:
AssertionError – if `obj` is not an instance of `cls`.

```
static assert_not_regex(text: str, unexpected_regex: str, message: str | None = None) →
None
```

Assert that `text` does not match regular expression `unexpected_regex`.

Parameters:

- **text** – a text to be matched
- **unexpected_regex** – a regular expression to be used
- **message=None** – the message to prepend to the error message
- **Raises** – AssertionError: if `text` matches regular expression `unexpected_regex`.

```
static assert_raises(expected_exception: Type[BaseException], callable_object: Any |
None = None, *args: Any, **kwargs: Any) → _AssertRaisesContext | None
```

Assert that `expected_exception` is raised. Can be used as a context manager or with a callable object that is run with `*args` and `**kwargs`.

Parameters:

- **expected_exception** – the expected exception that shall be raised
- **callable_object=None** – the callable object if not used as a context manager
- ***args** – variable positional arguments passed to `callable_object`
- ****kwargs** – variable keyword arguments passed to `callable_object`

Returns:
a context manager if `callable_object` is `None` else `None`.

```
static assert_raises_regex(expected_exception: Type[BaseException], expected_regex: str,
callable_object: Any | None = None, *args: Any, **kwargs: Any) → _AssertRaisesContext |
None
```

Assert that `expected_exception` is raised. Can be used as a context manager or with a callable object that is run with `*args` and `**kwargs`.

Parameters:

- **expected_exception** – the expected exception that shall be raised
- **expected_regex** – the regular expression that shall match the stringified representation of the exception
- **callable_object=None** – the callable object if not used as a context manager
- ***args** – variable positional arguments passed to `callable_object`
- ****kwargs** – variable keyword arguments passed to `callable_object`

Returns:
a context manager if `callable_object` is `None` else `None`.

```
static assert_regex(text: str, expected_regex: str, message: str | None = None) → None
```

Assert that `text` matches regular expression `expected_regex`.

Parameters:

- **text** – a text to be matched
- **expected_regex** – a regular expression to be used
- **message=None** – the message to prepend to the error message
- **Raises** – AssertionError: if `text` does not match regular expression `expected_regex`.

```
static assert_sequence_equal(a: Sequence, b: Sequence, message: str | None = None,
sequence_type: Type | None = None) → None
```

Assert that `a` equals `b` and that both are sequences.

Parameters:

- **a** – a sequence
- **b** – a sequence
- **sequence_type=None** – the sequence type to be used (if not `None`)
- **message=None** – the message to prepend to the error message

Raises:
AssertionError – if `a` and `b` are not equal.

```
static assert_set_equal(a: Set, b: Set, message: str | None = None) → None
```

Assert that `a` equals `b` and that both are of type `set`.

Parameters:

- **a** – a set
- **b** – a set
- **message=None** – the message to prepend to the error message

Raises:
AssertionError – if `a` and `b` are not equal.

```
static assert_str_equal(a: str, b: str, message: str | None = None) → None
```

Assert that `a` equals `b` and that both are of type `str`.

Parameters:

- **a** – a string
- **b** – a string
- **message=None** – the message to prepend to the error message

Raises:
AssertionError – if `a` and `b` are not equal.

```
static assert_true(expression: bool, message: str | None = None) → None
```

Assert that `expression` is `True`.

Parameters:

- **expression** – the boolean expression to be validated
- **message=None** – the message to prepend to the error message

Raises:
AssertionError – if `expression` is not `True`

```
static assert_tuple_equal(a: Tuple, b: Tuple, message: str | None = None) → None
```

Assert that `a` equals `b` and that both are of type `tuple`.

Parameters:

- **a** – a list
- **b** – a list
- **message=None** – the message to prepend to the error message

Raises:
AssertionError – if `a` and `b` are not equal.

```
attach_file(filename: str, title: str) → None
```

Attach a file to the current test that is put into the test report.

Parameters:

- **filename** – the filename to be attached
- **title** – the title

```
attach_url(url: str, title: str) → None
```

Attach an url to the current test that is put into the test report. This results in an external link and should be used for huge files.

Parameters:

- **url** – the url to be attached
- **title** – the title

```
delay(duration: float) → None
```

Delay execution for a given duration and print progress.

Parameters:
duration – sleep duration

```
expect(expected_value: Any, observed_value: Any, error_message: str | None = None) → None
```

Expect that `expectedValue` and `observedValue` are the same. If `expectedValue` and `observedValue` differ an `AssertionError` is raised.

Parameters:

- **expected_value** – the expected value.
- **observed_value** – the observed value that is compared to `expectedValue`.
- **errorMessage=None** – an optional message.

```
expect_step(title: str, expected_value: Any, observed_value: Any, error_message: str |
None = None) → None
```

Creates a test step in test report and on stdout using `title`. Expect that *expectedValue* and *observedValue* are the same. If `expectedValue` and `observedValue` differ an `AssertionError` is raised.

Parameters:

- **title** – the step's title.
- **expected_value** – the value that is expected.
- **observed_value** – the current observed value that is compared to `expectedValue`.
- **errorMessage=None** – an optional message.

```
run_background(func: Callable[[...], Any], name: str | None = None, stop:
Callable[[...], bool] | None = None, force_stop: bool = True, join_timeout: float = 1.0,
args: Tuple[Any] | None = None, kwargs: Dict[str, Any] | None = None) → Callable[[], None]
```

With this method you can run another method in background.

Parameters:

- **func** – the method to be run in background.
- **name=None** – a name for the background task. Default: `None`. If *name* is `None` *func.__name__* is used.
- **stop=None** – callable to stop the thread
- **force_stop=True** – force the thread to stop when leaving its scope (can impact performance)
- **join_timeout=1** – the timeout to be used when calling `join` on the thread.
- **args=None** – a tuple of positional arguments that are passed to the background task when it is called.
- **kwargs=None** – a dictionary of keyword arguments that are passed to the background task when it is called.

Returns:
a callable object is returned that stops

execution when called.

Return type:
stop-trigger

```
run_periodic(func: Callable[[...], Any], period: float, maximum_period: float | None =
None, name: str | None = None, raise_exception: bool = True, args: Tuple[Any] | None =
None, kwargs: Dict[str, Any] | None = None) → Callable[[], None]
```

With this method you can run another method periodically in background with a given period.

Parameters:

- **func** – the method to be called periodically in background.
- **period** – the period in second.
- **maximum_period=None** – if set to a float >= `period` `func` may take up to `maximum_period` time without raising an exception. A warning is printed out to stdout instead.
- **name=None** – a name for the background task.
- **raise_exceptions=True** – if set to `True` exceptions are put into the exception queue to let a test fail if a background thread fails.
- **args=None** – a tuple of positional arguments that are passed to the background task when it is called.
- **kwargs=None** – a dictionary of keyword arguments that are passed to the background task when it is called.

Returns:
a callable object is returned that stops

execution when called.

Return type:
stop-trigger

```
set_metadata(name: str, value: Any) → None
```

Set metadata for a running test.

Parameters:

- **name** – the name of the metadata
- **value** – the value of the metadata (`value` is casted to a string automatically)

Raises:
AssertionError – if `name` was already set to non accidentally overwrite existing metadata

```
set_output_capture_enable(enabled: bool = True) → None
```

Enables or disables capturing of stdout and stderr for later use in test reports.

Parameters:
enabled=True – the new enabled state

```
set_up() → None
```

This method is called from test runner before a test is executed to create the test-fixtures.

```
classmethod set_up_class() → None
```

This method is called for each class instance.

```
skip_step(message: str) → None
```

Skip the current test step.

```
skip_test(message: str) → None
```

Skip the current test.

```
async sleep(duration: float) → None
```

Delay execution (async) for a given duration and print progress.

Parameters:
duration – sleep duration

```
step(title: str, expected_result: str | None = None, *args: Any, **kwargs: Any) →
TestStep
```

Run a test step that is documented in the test report.

Parameters:

- **title** – the step's title.
- ***args** – a tuple of positional arguments for the step's context.
- ****kwargs** – a dictionary of keyword arguments for the step's context.

```
stop_thread(name_or_function: str | Callable) → int
```

This method stops a specific thread with the name or function described by `name_or_function` by calling `join` with join timeout supplied by `run_periodic` or `run_background` or on it. It does not set a stop condition so the thread must end and must not include an endless loop. If the thread is still alive after the first join messages are printed to inform the user.

Parameters:
name_or_function – the name or the function (reference) of the thread to be stopped.
Returns:
the number of stopped threads.
Return type:
int
Raises:
Exception – if the thread was not found

```
tear_down() → None
```

This method is called from test runner after a test was executed regardless of its result.

This method shall destroy all used fixtures.

```
classmethod tear_down_class() → None
```

This method is called for each class instance.

Fixtures

Fixtures provide data, objects or services for tests. They are automatically initialized and uninitialized by the test runner.

Fixtures have a scope:

- `'test'`: Fixture is initialized and uninitialized for every single test. Smallest scope.

- `'class'`: Fixture is initialized when a class holding tests is run for the first time, and uninitialized after all tests of the are run.
- `'module'`: Fixture is initialized once for all tests in a module.
- `'session'`: Fixture is initialized for the whole session. Greatest scope.

Fixtures are implemented as python functions returning an object or python generators yielding a single object. Python generators offer set up and tear down code while python function fixture only offer set up code.

To use a fixture in a test, simply add a parameter with the fixture's name to the test. The yielded object is passed to the test when it is executed by the test runner.

Fixtures are collected automatically by the test runner by searching the test specifiers.

To create a fixture decorate it with `htf.fixture`.

```
@htf.fixture('test')
def server() -> Generator["Server", None, None]:
    server = Server()
    yield server
    server.stop()

# pass a parameter with the fixture's name to use the fixture
def test_server(server):
    server.handle_request()
```

Fixtures can also depend on other fixtures. To create a dependency simply add a parameter to the fixture like you would for a test.

```
@htf.fixture('test')
def client(server):
    return Client()

def test_client(client, server):
    server.run()
    client.connect()
```

Circular dependencies are detected and raise an exception.

Requested fixtures must have the same or a greater scope than the requesting fixture. I.e. a fixture with scope 'test' can request fixtures with all other scopes, but a fixture with scope 'session' can only request other fixtures with 'session' scope.

You can change the fixture name by setting the `name` parameter.

Generally fixtures are only called when specifically requested by tests or other fixtures. For auxillary fixtures which require no direct interaction, you can set the `auto` parameter to `True`. In this case they are set up and torn down when their scope is entered or left.

Fixtures can be async, too.

```
@htf.fixture('test')
async def async_fixture() -> AsyncGenerator["Server", None]:
    server = await Server.create()
    yield server
    await server.stop()
```

Creating Fixtures

To create fixtures develop them using the `htf.fixture` decorator.

Using Fixtures

To use fixtures, add the sources containing the fixtures as test specifier to the `htf` command line utility or `htf.main`. The fixtures are collected automatically.

Fixture Decorator

The `htf.fixture` decorator is used to create fixtures.

```
htf.fixture(scope: str, name: str | None = None, auto: bool = False) → Callable[[...], Any]
```

Decorator to mark a function as a fixture.

Parameters:

- **scope** – the scope determining the life-time of the fixture. Choose out of ‘session’, ‘module’, ‘class’ and ‘test’.
- **name=None** – the name to be used for the fixture
- **auto=False** – automatically call the fixture when its scope is entered, instead of when requested

Built-in Fixtures

htf is shipped with many built-in fixtures for different use cases.

Assertions (assertions)

`assertions` lets you easily do assertions. The fixture is `htf.assertions`. The scope is `'session'`.

```
def test_with_assertions(assertions):
    """
    Args:
        assertions (htf.assertions): the assertions fixture
    """
    assertions.assert_true(True)
```

File and URL Attachments (attachments)

`htf.fixtures.attachments` lets you attach files and URLs to a test report. The scope is `'test'`.

```
def test_attach_file(attachments):
    """
    Args:
        attachments (htf.fixtures.attachments): the attachments fixture
    """
    attachments.attach_file("example.jpg", "an example image attached to a report")
```

Attaching files as bytes

```
def test_attach_file_as_bytes(attachments):
    """
    Args:
        attachments (htf.fixtures.attachments): the attachments fixture
    """
    filename = "example.png"
    mime_type = "image/png"
    with open(filename, "rb") as f:
        data_bytes = f.read()
        attachments.attach_bytes("Example", data_bytes, filename, mime_type)
```

Attaching a URL

```
def test_attach_url(attachments):
    """
    Args:
        attachments (htf.fixtures.attachments): the attachments fixture
    """
    attachments.attach_url("https://hilster.io", "an example URL attached to a report")
```

```
class htf.fixtures.attachments(result: TestResult)
```

Attachments to the test report. Scope is `'test'`.

```
attach_bytes(title: str, data_bytes: bytes, filename: str, mime_type: str) → None
```

Accepts bytes and mime type and converts them to a data url and attaches it to the result.

Parameters:

- **title** – the title
- **data_bytes** – the bytes to attach
- **filename** – the file name
- **mime_type** – MIME type of the data bytes

```
attach_file(filename: str, title: str) → None
```

Attach a file to the current test that is put into the test report.

Parameters:

- **filename** – the filename to be attached
- **title** – the title

```
attach_url(url: str, title: str) → None
```

Attach an url to the current test that is put into the test report. This results in an external link and should be used for huge files.

Parameters:

- **url** – the url to be attached
- **title** – the title

```
set_maximum_attachment_size(size: int) → None
```

Set the maximum attachment size to `size` bytes.

Parameters:
size – the maximum attachment size in bytes. Default ist 10 MB.

To increase the maximum attachment size simply create a new fixture with the maximum size set to the desired value:

```
@htf.fixture('test')
def big_attachments(attachments):
    attachments.set_maximum_attachment_size(100*1024*1024)  # 100 MB
    yield attachments
```

Context (context)

The `htf.fixtures.context` is used to share data between steps within a behavior driven test. The scope is `'test'`.

```
class htf.fixtures.context
```

A context fixture used to share context data in behavior driven steps.

```
@htf.given("the DUT is up and running")
def up_and_running(context):
    context.dut = DUT()
```

```
@htf.when("button is pressed")
def button_is_pressed(context):
    context.dut.press_button()

@htf.then(r"the button state shall be '([a-z]+)'")
def button_state(pressed, context):
    htf.assert_equal(context.dut.get_button_state(), pressed)
```

Delay Functions (delay)

`htf.fixtures.delay` lets you delay tests. The remaining delay is printed to the terminal. `htf.fixtures.delay` is callable. The scope is `'session'`.

```
def test_with_delay(delay):
    """
    Args:
        delay (htf.fixtures.delay): the delay fixture
    """
    delay(5)
```

```
class htf.fixtures.delay
```

Delay execution for a given duration and print progress. Scope is `'session'`.

Parameters:
duration – delay duration in seconds

Sleep Functions (async) (sleep)

`htf.fixtures.sleep` lets you delay tests. The remaining delay is printed to the terminal. `htf.fixtures.sleep` is callable. The scope is `'session'`.

```
async def test_with_sleep(desleeplay):
    """
    Args:
        sleep (htf.fixtures.sleep): the sleep fixture
    """
    await sleep(5)
```

```
class htf.fixtures.sleep
```

Delay execution for a given duration and print progress. Used in async context only. Scope is `'session'`.

Parameters:
duration – delay duration in seconds

Interactive Testing (interaction)

The `htf.fixtures.interaction` fixture is documented in the Interactive Testing section.

Issue Tracking (jira)

The usage of the `htf.fixtures.jira` fixture is documented in the JIRA (Atlassian) section.

Append Metadata to Test Reports (metadata)

`htf.fixtures.metadata` lets you add metadata to tests. `htf.fixtures.metadata` works like a `dict` but also support setters and getters. The scope is `'test'`.

```
def test_with_metadata(metadata):
    """
    Args:
        metadata (htf.fixtures.metadata): the metadata fixture
    """
    metadata.set("key", "value")
    metadata["Date"] = datetime.now()
```

```
class htf.fixtures.metadata(result: TestResult)
```

Set metadata in test report. Scope is `'test'`.

```
get(name: str) → Any
```

Get metadata for name.

Parameters:
name – metadata name
Returns:
the value
Return type:
object

```
set(name: str, value: Any) → None
```

Set a metadata name to a value.

Parameters:

- **name** – metadata name
- **value** – metadata value

Access Settings (settings)

`Settings` lets you easily access settings in a test. The returned object is an instance of `htf.Settings`. The scope is `'session'`.

```
def test_with_settings(settings):
    """
    Args:
        settings (htf.fixtures.settings): the settings fixture
    """
    print(settings.example_setting)
```

Access Testrun Parameters (parameters)

`Parameters` lets you easily access test run parameters in a test. The returned object is an instance of a read-only dict like object. The scope is `'session'`.

```
def test_parameters(parameters):
    """
    Args:
        settings (htf.fixtures.parameters): the parameters fixture
    """
    print(parameters['name'])

   # htf --parameter name=foobar
   # htf.main(parameters=dict(name="foobar"))
```

Parameters can also be used to control the behavior of other fixtures.

```
@htf.fixture(scope="test")
def io(parameters):
    if parameters.get('io_enable', default=False):
        yield IO(parameters['io_settings'])
    else:
        yield SimulatedIO(parameters['io_settings'])
```

```
class htf.fixtures.Parameters(data: Dict[Any, Any])
```

Access parameters in test report. Scope is `'session'`. Is instantiated automatically.

Structure Tests (step)

`htf.fixtures.step` lets you separate your test into steps. The scope is `'test'`.

```
def test_with_step(step):
    """
    Args:
        step (htf.fixtures.step): the step fixture
    """
    with step("First step"):
        htf.assert_true(True)

    # steps can be nested, too
    with step("Outer step"):
        with step("Inner step"):
            htf.assert_true(True)

    # steps can have an expected result
    with step("Succeeding step", expected_result="this step succeeds"):
        htf.assert_true(True)
```

You can also add additional data to steps that will be put into the test report.

```
def test_with_additional_step_data(step):
    """
    Args:
        step (htf.fixtures.step): the step fixture
    """
    with step("First step", 1, 2, 3, step_id=100):
        htf.assert_true(True)
```

To skip a step run

```
def test_with_skipped_step(step):
    """
    Args:
        step (htf.fixtures.step): the step fixture
    """
    with step("Skipped step"):
        step.skip("This step is skipped.")
```

`htf.fixtures.step` supports expected and actual results, too.

```
def test_with_expected_and_actual_result(step):
    """
    Args:
        step (htf.fixtures.step): the step fixture
    """
    with step("Step with expected and actual result", expected_result="The expected result") as
current_step:
        current_step.set_actual_result("the actual result")
```

`htf.fixtures.step` can be used as a `dict` to add data to the test result at runtime. The default contents are read-only.

```
def test_with_expected_and_actual_result(step):
    """
    Args:
        step (htf.fixtures.step): the step fixture
    """
    with step("Step with expected and actual result", expected_result="The expected result") as
current_step:
        current_step["actual_result"] = "the actual result"
```

```
class htf.fixtures.step(result: TestResult)
```

Run a test step that is documented in the test report. Scope is `'test'`.

Parameters:

- **title** – the step's title.
- **callable_obj=None** – a callable object that is called within the step or `None`.
- ***args** – a tuple of positional arguments for the step's context.
- ****kwargs** – a dictionary of keyword arguments for the step's context.

```
skip(reason: str | None = None) → None
```

Skip a step.

Parameters:
reason=None – the reason for the skipped step
Raises:
htf.SkipStep –

Test Information (test)

`htf.fixtures.test` gives you access to information about the current test. The scope is `'test'`.

```
def test_info(test):
    """
    Args:
        test (htf.fixtures.test): the test fixture
    """
    print("Test function name:", test.name)
    print("Test class name:", test.class_name)
    print("Test module:", test.module)
    print("Docstring:", test.doc)
    print("Tags:", test.tags)
    print("Current stdout:", test.stdout)
    print("Current stderr:", test.stderr)
    print("Current result:", test.result)  # None
```

The properties `test.result`, `test.was_successful`, `test.was_not_successful`, `test.was_failure`, `test.was_error`, `test.exception_type`, `test.exception_value` and `test.exception_traceback` are only available after the test was run in the context of another fixture using the `test` fixture.

Thus, it is possible to add actions in fixtures depending on the test's result.

```
@htf.fixture(scope="test", auto=True)
def after_failed_test(test):
    yield

    if test.was_not_successful:
        print(f"The test {test.name} was not successful! It failed with
{test.exception_type.__class__.__name__}: '{test.exception_value}'")


def test_failure(test):
    assert False, "This test fails"
```

```
class htf.fixtures.test(result: TestResult)
```

Information about the running test. Scope is `'test'`.

Feature Information (feature)

`htf.fixtures.feature` gives you access to information about the current feature running in a behavior driven test. The scope is `'test'`.

```
@htf.given("when I would like to see feature information")
def given_feature(feature):
    """
    Args:
        test (htf.fixtures.feature): the feature fixture
    """
    print("Feature name:", feature.name)
    print("Feature doc:", feature.doc)
    print("Feature filename:", feature.filename)
```

```
class htf.fixtures.feature(result: TestResult)
```

Information about the running feature in behavior driven testing. Scope is `'test'`.

Scenario Information (scenario)

`htf.fixtures.scenario` gives you access to information about the current scenario running in a behavior driven test. The scope is `'test'`.

```
@htf.given("when I would like to see scenario information")
def given_feature(scenario):
    """
    Args:
        test (htf.fixtures.scenario): the scenario fixture
    """
    print("Scenario name:", scenario.name)
```

```
class htf.fixtures.scenario(result: TestResult)
```

Information about the running scenario in behavior driven testing. Scope is `'test'`.

Statistics (statistics)

`htf.fixtures.statistics` gives you statistics about the current test session. The scope is `'session'`.

```
def test_statistics(statistics):
    """
    Args:
        statistics (htf.fixtures.statistics): the statistics fixture
    """
    print("Total number of tests:", statistics.number_of_tests)
    print("Tests run so far:", statistics.tests_run)
    print("Percent of tests finished:", statistics.done_percent)
    print("Successes:", statistics.successes)
    print("Failures:", statistics.failures)
    print("Errors:", statistics.errors)
    print("Skipped:", statistics.skipped)
    print("Percent of successes:", statistics.success_percent)
    print("Percent of failures:", statistics.failure_percent)
    print("Percent of errors:", statistics.error_percent)
    print("Percent of skipped:", statistics.skipped_percent)
```

```
class htf.fixtures.statistics(statistics_provider: statistics_provider)
```

Statistics about the test session. Scope is `'session'`.

Run Threads (threads)

`htf.fixtures.threads` lets you start threads that are stopped automatically. You can start continuously running threads and periodically executed threads. The scope is `'test'`. All started threads are automatically stopped after the starting test ends. You do not need to worry about stopping threads yourself.

```
def test_with_background_thread(threads):
    """
    Args:
```

```
        threads (htf.fixtures.threads): the threads fixture
    """
    def background_thread():
        while True:
            print("running in background")

    threads.run_background(background_thread)

def test_with_periodic_thread(threads):
    """
    Args:
        threads (htf.fixtures.threads): the threads fixture
    """
    def periodic_thread():
        print("running periodically in background")

    threads.run_periodic(periodic_thread, period=1.0)
    time.sleep(5)
```

Automatic stoppable threads are instrumented to be stoppable under all circumstances. Thus the performance may be impacted but `while True` loops are allowed in these threads, the development is very easy and you do not need any overhead to stop the thread. If you need more performance for background threads you have to make your threads stoppable from the outside and not use `while True` loops.

```
def test_stoppable_thread(threads):
    # create an event to stop the thread
    stop_condition = threading.Event()

    def background_thread():
        while not stop_condition.is_set():
            print("running ..")
            time.sleep(.5)

    # normally started threads that is stopped automatically with reduced performance
    threads.run_background(background_thread)

    # high performance background thread that
    stop = threads.run_background(background_thread, stop=stop_condition.set, force_stop=False)

    # wait
    time.sleep(2.0)

    # stop the thread explicitly
    stop()
```

```
class htf.fixtures.threads
```

Fixture for threads with scope ‘test’ that are stopped automatically and can run in background and periodically.

```
run_background(func: Callable[[...], Any], name: str | None = None, stop:
Callable[[...], bool] | None = None, force_stop: bool = True, join_timeout: float = 1.0,
args: Tuple[Any] | None = None, kwargs: Dict[str, Any] | None = None) → Callable[[], None]
```

With this method you can run another method in background.

Parameters:

- **func** – the method to be run in background.
- **name=None** – a name for the background task. Default: `None`. If *name* is `None` *func.__name__* is used.
- **stop=None** – callable to stop the thread
- **force_stop=True** – force the thread to stop when leaving its scope (can impact performance)
- **join_timeout=1** – the timeout to be used when calling `join` on the thread.
- **args=None** – a tuple of positional arguments that are passed to the background task when it is called.
- **kwargs=None** – a dictionary of keyword arguments that are passed to the background task when it is called.

Returns:
a callable object is returned that stops

execution when called.

Return type:
stop-trigger

```
run_periodic(func: Callable[[...], Any], period: float, maximum_period: float | None =
None, name: str | None = None, raise_exception: bool = True, args: Tuple[Any] | None =
None, kwargs: Dict[str, Any] | None = None) → Callable[[], None]
```

With this method you can run another method periodically in background with a given period.

Parameters:

- **func** – the method to be called periodically in background.
- **period** – the period in second.
- **maximum_period=None** – if set to a float >= `period` `func` may take up to `maximum_period` time without raising an exception. A warning is printed out to stdout instead.
- **name=None** – a name for the background task.
- **raise_exceptions=True** – if set to `True` exceptions are put into the exception queue to let a test fail if a background thread fails.
- **args=None** – a tuple of positional arguments that are passed to the background task when it is called.
- **kwargs=None** – a dictionary of keyword arguments that are passed to the background task when it is called.

Returns:
a callable object is returned that stops

execution when called.

Return type:
stop-trigger

```
stop() → None
```

This method stops all background threads by calling `thread.join()` until it is not alive anymore.

```
stop_thread(name_or_function: str | Callable) → int
```

This method stops a specific thread with the name or function described by `name_or_function` by calling `join` with join timeout supplied by `run_periodic` or `run_background` or on it. It does not set a stop condition so the thread must end and must not include an endless loop. If the thread is still alive after the first join messages are printed to inform the user.

Parameters:
name_or_function – the name or the function (reference) of the thread to be stopped.
Returns:
the number of stopped threads.
Return type:
int
Raises:
Exception – if the thread was not found

Run Session Threads (session_threads)

Like `htf.fixtures.threads` but with scope `'session'`.

Run Processes (processes)

`htf.fixtures.processes` lets you start processes that are stopped automatically. You can start single background processes and process pools. It is based on `multiprocessing`. The scope is `'test'`. All started processes are automatically stopped after the starting test ends. You do not need to worry about stopping processes yourself.

Create a single background process:

```
def test_process_stopping(processes):
    """
    Test a stopping background process.

    Args:
        processes (htf.fixtures.processes): the processes fixture
    """
    def process(*what):
        print("process is running ..")
        for i in range(2):
            print(" ".join(what))
            time.sleep(1.0)

    processes.run(target=process, args=("a", "b"))
    time.sleep(3.0)
```

Create a pool and run tasks in parallel:

```
def square(x):
    return x**2


def test_pool(processes):
    """
    Test processes fixture's pool.

    Args:
        processes (htf.fixtures.processes): the processes fixture
    """
    pool = processes.pool(processes=2)
    print(pool.map(square, range(10)))
    time.sleep(1.0)
```

```
class htf.fixtures.processes
```

Fixture for background processes with scope `'test'` that are stopped automatically. Uses `multiprocessing`.

```
pool(processes: int | None = None, initializer: Callable[[...], Any] | None = None,
initargs: Iterable[Any] = (), maxtasksperchild: int | None = None, timeout: float = 1.0) →
multiprocessing.pool.Pool
```

Create a pool to run tasks in parallel.

Parameters:

- **processes=None** – the number of child processes to be started.
- **initializer=None** – f initializer is not None then each worker process will call `initializer(*initargs)` when it starts.
- **initargs=()** – arguments for `initializer`.
- **maxtasksperchild=None** – the maximum number of tasks per child process.
- **timeout=1.0** – the timeout for joining the pool process.

Returns:
the pool

```
run(group: None = None, target: Callable[[...], Any] | None = None, name: str | None =
None, args: Iterable[Any] = (), kwargs: Dict[str, Any] | None = None, daemon: bool | None
= None, timeout: float = 1.0) → Process
```

Run a background process.

Parameters:

- **group=None** – the group

- **target=None** – A callable to be run in background. Must be pickleable.
- **name=None** – the name of the background process.
- **args=()** – the positional arguments passed to the background process callable.
- **kwargs=None** – the keyword arguments passed to the background process callable.
- **daemon=None** – if set to `True` the child process is terminated if the parent process ends.
- **timeout=1.0** – the timeout for joining, terminating and killing the process.

Returns:
the process

Run Session Processes (session_processes)

Like `htf.fixtures.processes` but with scope `'session'`.

Type Hints for fixtures

To be able to use code completion in you IDE add type hints or docstrings to your tests so that the IDE is able to find the right fixture.

```
# type hint:
def test_step(step : htf.fixtures.step):
    with step("A step"):
        pass

# docstring:
def test_step(step):
    """
    This test ...

    Args:
        step (htf.fixtures.step): the step fixture
    """
    with step("A step"):
        pass
```

Example: Cache Fixture

Assume you want to cache data over multiple test runs to reduce the runtime. You could create a cache fixture.

```
import json
# cache fixture
@htf.fixture('session')
def cache():
    # set up
    filename = "cache.json"
    try:
        with open(filename, "r") as fp:
            data = json.load(fp)
    except FileNotFoundError:
        with open(filename, "w") as fp:
```

```
            json.dump({}, fp)
            print("Created cache file '{}'".format(filename))
        data = {}

    yield data

    # tear down
    with open(filename, "w") as fp:
        json.dump(data, fp)
        print("Wrote cache file '{}'".format(filename))

# test using cache fixture
def test_cache(cache):
    try:
        # count up test run number
        cache["#"] += 1
    except KeyError:
        cache["#"] = 1

    print("This is test run {}".format(cache["#"]))
```

Tagging Fixtures

Fixtures can also be tagged. Thus it is easy to select a set of fixture with a tag expression.

For example you can create fixtures for simulated and hardware in the loop devices with different tags and select a group to run the same tests with and without hardware.

`htf` support `-F <expression>` and `--fixture-tags=<expression>`. Unlike selecting tests by tags all untagged fixtures are included in the result. So if you use fixture tags they will only be used to filter actually tagged fixtures.

Example:

```
class Device:
    pass


@htf.tags("simulation")
@htf.fixture('test', name="device")
def device_simulation():
    print("simulated device")
    yield Device()


@htf.tags("hardware")
@htf.fixture('test', name="device")
def device_hardware():
    print("hardware device")
    yield Device()


def test_device(device):
    pass
```

```
if __name__ == "__main__":
    htf.main(fixture_tags="hardware")
    # $ htf --fixture-tags=hardware
```

Accessing Fixtures directly

You might need to access one fixture in a location where it is not already set up. This is useful for fixtures that are not part of the argument list of the test function. To do that, you can use `htf.get_fixture` (async variant) or `htf.get_fixture_sync` (sync variant).

If the fixture was not instantiated before, then it will be instantiated as soon as the upper functions are called.

For async (coroutine) fixtures, the following example shows the behavior.

```
@htf.fixture('test')
async def example_fixture():
    yield "Example"


async def test_device():
    fixture = await htf.get_fixture("example_fixture")
```

In case you don't want to use coroutines, it is still possible to use the sync variant.

```
@htf.fixture('test')
def example_fixture():
    yield "Example"


def test_device():
    fixture = htf.get_fixture_sync("example_fixture")
```

Note

`htf.get_fixture_sync` will raise an exception when trying to instantiate an async fixture. It will however return already instantiated fixtures regardless of its type.

```
async htf.get_fixture(name: str) → Any
```

Returns a fixture object as identified by the name parameter. If a fixture requires other fixtures, they will be instantiated, if necessary. This is the async variant which returns a coroutine, so ideal for async fixtures.

Parameters:
name – name of the requested fixture
Returns:
fixture instance

Raises:
Exception – if a circular dependency is detected during resolution

```
htf.get_fixture_sync(name: str) → Any
```

Returns a fixture object as identified by the name parameter. If a fixture requires other fixtures, they will be instantiated, if necessary. This is the sync variant which returns the fixture instance.

Parameters:
name – name of the requested fixture
Returns:
fixture instance
Raises:

- **Exception** – if a circular dependency is detected during resolution
- **TypeError** – if the instantiated fixture is a coroutine

Assertions

Assertions are used to assert conditions of your tests. With assertions the actual testing is realized.

Assert Statement

The easiest way to assert a statement is to use the `assert` statement.

```
def test_with_assert():
    assert True, "This is the assertions failure message"
```

Built-in Assertions

To write tests with finegrained assertions you can use the built-in assertions.

All specialized assertions can be used via the module `htf.assertions` or the `assertions` fixture.

```
def test_with_assertions_fixture(assertions):
    assertions.assert_true(True, "This is the assertions failure message")
    assertions.assert_almost_equal(1.00, 1.01, places=2)
    # etc.


def test_with_assertions_module():
    htf.assertions.assert_true(True, "This is the assertions failure message")
    htf.assertions.assert_almost_equal(1.00, 1.01, places=2)
```

```
htf.assertions.assert_almost_equal(a: float, b: float, places: int | None = None, delta:
float | None = None, message: str | None = None) → None
```

Assert that `a` and `b` are almost equal. Useful to compare floating point numbers. Comparison is realized either using a maximum `delta` or the number of decimal `places`

Parameters:

- **a** – number
- **b** – number
- **places=None** – the number of decimal places for comparison. If `None` `7` decimal places are used.
- **delta=None** – the maximum delta
- **message** – the message to prepend to the error message

Raises:
AssertionError – if `a` and `b` are not almost equal

```
htf.assertions.assert_dict_equal(a: Dict, b: Dict, message: str | None = None) → None
```

Assert that `a` equals `b` and that both are of type `dict`.

Parameters:

- **a** – a dict
- **b** – a dict
- **message** – the message to prepend to the error message

Raises:
AssertionError – if `a` and `b` are not equal

```
htf.assertions.assert_equal(a: Any, b: Any, message: str | None = None) → None
```

Assert that `a` equals `b`. Tries to find a registered equality asserter and uses the base assert if no special assert is registered.

Parameters:

- **a** – an object
- **b** – an object
- **message** – the message to prepend to the error message

Raises:
AssertionError – if `a` and `b` are not equal

```
htf.assertions.assert_false(expression: bool, message: str | None = None) → None
```

Assert that `expression` is `False`.

Parameters:

- **expression** – the boolean expression to be validated

- **message** – the message to prepend to the error message

Raises:
AssertionError – if `expression` is not `False`

```
htf.assertions.assert_greater(a: Any, b: Any, message: str | None = None) → None
```

Assert that `a` is greater than `b`.

Parameters:

- **a** – a comparable object
- **b** – a comparable object
- **message** – the message to prepend to the error message

Raises:
AssertionError – if `a` is not greater than `b`

```
htf.assertions.assert_greater_equal(a: Any, b: Any, message: str | None = None) → None
```

Assert that `a` is greater or equal to `b`.

Parameters:

- **a** – a comparable object
- **b** – a comparable object
- **message** – the message to prepend to the error message

Raises:
AssertionError – if `a` is not greater or equal to `b`

```
htf.assertions.assert_in(member: Any, container: Iterable, message: str | None = None) →
None
```

Assert that `container` contains `member`.

Parameters:

- **member** – the object that shall be included in `container`
- **container** – an iterable to search for `member`
- **message** – the message to prepend to the error message

Raises:
AssertionError – if `member` is not included in `container`

```
htf.assertions.assert_is(a: Any, b: Any, message: str | None = None) → None
```

Assert that `id(a)` equals `id(b)`.

Parameters:

- **a** – an object

- **b** – another object
- **message** – the message to prepend to the error message

Raises:
AssertionError – if `id(a)` does not equal `id(b)`

```
htf.assertions.assert_is_instance(obj: Any, cls: Type, message: str | None = None) → None
```

Assert that `obj` is an instance of `cls`.

Parameters:

- **obj** – an instance
- **cls** – a class `obj` shall be an instance of
- **message** – the message to prepend to the error message

Raises:
AssertionError – if `obj` is not an instance of `cls`

```
htf.assertions.assert_is_none(obj: Any, message: str | None = None) → None
```

Assert that `obj` is `None`.

Parameters:

- **obj** – an object
- **message** – the message to prepend to the error message

Raises:
AssertionError – if `obj` is not `None`

```
htf.assertions.assert_is_not(a: Any, b: Any, message: str | None = None) → None
```

Assert that `id(a)` does not equal `id(b)`.

Parameters:

- **a** – an object
- **b** – another object
- **message** – the message to prepend to the error message

Raises:
AssertionError – if `id(a)` equals `id(b)`

```
htf.assertions.assert_is_not_instance(obj: Any, cls: Type, message: str | None = None) →
None
```

Assert that `obj` is not an instance of `cls`.

Parameters:

- **obj** – an instance

- **cls** – a class `obj` shall not be an instance of
- **message** – the message to prepend to the error message

Raises:
AssertionError – if `obj` is an instance of `cls`

```
htf.assertions.assert_is_not_none(obj: Any, message: str | None = None) → None
```

Assert that `obj` is not `None`.

Parameters:

- **obj** – an object
- **message** – the message to prepend to the error message

Raises:
AssertionError – if `obj` is `None`

```
htf.assertions.assert_less(a: Any, b: Any, message: str | None = None) → None
```

Assert that `a` is less than `b`.

Parameters:

- **a** – a comparable object
- **b** – a comparable object
- **message** – the message to prepend to the error message

Raises:
AssertionError – if `a` is not less than `b`

```
htf.assertions.assert_less_equal(a: Any, b: Any, message: str | None = None) → None
```

Assert that `a` is less or equal to `b`.

Parameters:

- **a** – a comparable object
- **b** – a comparable object
- **message** – the message to prepend to the error message

Raises:
AssertionError – if `a` is not less or equal to `b`

```
htf.assertions.assert_list_equal(a: List, b: List, message: str | None = None) → None
```

Assert that `a` equals `b` and that both are of type `list`.

Parameters:

- **a** – a list
- **b** – a list

- **message** – the message to prepend to the error message

Raises:
AssertionError – if `a` and `b` are not equal or not lists

```
htf.assertions.assert_multi_line_equal(a: str, b: str, message: str | None = None) → None
```

Assert that `a` equals `b` and that both are of type `str`.

Parameters:

- **a** – a string
- **b** – a string
- **message** – the message to prepend to the error message

Raises:
AssertionError – if `a` and `b` are not equal or not strings

```
htf.assertions.assert_not_almost_equal(a: float, b: float, places: int | None = None,
delta: float | None = None, message: str | None = None) → None
```

Assert that `a` and `b` are not almost equal. Useful to compare floating point numbers. Comparison is realized either using a maximum `delta` or the number of decimal `places`

Parameters:

- **a** – number
- **b** – number
- **places=None** – the number of decimal places for comparison. If `None` `7` decimal places are used.
- **delta=None** – the maximum delta
- **message** – the message to prepend to the error message

Raises:
AssertionError – if `a` and `b` are almost equal

```
htf.assertions.assert_not_equal(a: Any, b: Any, message: str | None = None) → None
```

Assert that `a` does not equal `b`.

Parameters:

- **a** – an object
- **b** – an object
- **message** – the message to prepend to the error message

Raises:
AssertionError – if `a` and `b` are equal

```
htf.assertions.assert_not_in(member: Any, container: Iterable, message: str | None = None) →
None
```

Assert that `container` does not contain `member`.

Parameters:

- **member** – the object that shall not be included in `container`
- **container** – an iterable to search for `member`
- **message** – the message to prepend to the error message

Raises:
AssertionError – if `member` is included in `container`

```
htf.assertions.assert_not_is_instance(obj: Any, cls: Type, message: str | None = None) →
None
```

Assert that `obj` is not an instance of `cls`.

Parameters:

- **obj** – an instance
- **cls** – a class `obj` shall not be an instance of
- **message** – the message to prepend to the error message

Raises:
AssertionError – if `obj` is an instance of `cls`

```
htf.assertions.assert_not_regex(text: str, unexpected_regex: str,
message: str | None = None) → None
```

Assert that `text` does not match regular expression `unexpected_regex`.

Parameters:

- **text** – a text to be matched
- **unexpected_regex** – a regular expression to be used
- **message** – the message to prepend to the error message

Raises:
AssertionError – if `text` matches regular expression `unexpected_regex`

```
htf.assertions.assert_raises(expected_exception: Type[BaseException], callable_object: Any |
None = None, *args: Any, **kwargs: Any) → _AssertRaisesContext | None
```

Assert that `expected_exception` is raised. Can be used as a context manager or with a callable object that is run with `*args` and `**kwargs`.

Parameters:

- **expected_exception** – the expected exception that shall be raised
- **callable_object=None** – the callable object if not used as a context manager
- ***args** – variable positional arguments passed to `callable_object`
- ****kwargs** – variable keyword arguments passed to `callable_object`

Returns:
a context manager if `callable_object` is `None` else `None`.
Raises:
AssertionError – if `expected_exception` is not raised

```
htf.assertions.assert_raises_regex(expected_exception: Type[BaseException], expected_regex:
str, callable_object: Any | None = None, *args: Any, **kwargs: Any) → _AssertRaisesContext |
None
```

Assert that `expected_exception` is raised. Can be used as a context manager or with a callable object that is run with `*args` and `**kwargs`.

Parameters:

- **expected_exception** – the expected exception that shall be raised
- **expected_regex** – the regular expression that shall match the stringified representation of the exception
- **callable_object=None** – the callable object if not used as a context manager
- ***args** – variable positional arguments passed to `callable_object`
- ****kwargs** – variable keyword arguments passed to `callable_object`

Returns:
a context manager if `callable_object` is `None` else `None`.
Raises:
AssertionError – if `expected_exception` is not raised or does not match `expected_regex`

```
htf.assertions.assert_regex(text: str, expected_regex: str, message: str | None = None) →
None
```

Assert that `text` matches regular expression `expected_regex`.

Parameters:

- **text** – a text to be matched
- **expected_regex** – a regular expression to be used
- **message** – the message to prepend to the error message

Raises:
AssertionError – if `text` does not match regular expression `expected_regex`

```
htf.assertions.assert_sequence_equal(a: Sequence, b: Sequence, message: str | None = None,
sequence_type: Type | None = None) → None
```

Assert that `a` equals `b` and that both are sequences.

Parameters:

- **a** – a sequence
- **b** – a sequence
- **message** – the message to prepend to the error message
- **sequence_type=None** – the sequence type to be used (if not `None`)

Raises:
AssertionError – if `a` and `b` are not equal or don't match the `sequence_type`

```
htf.assertions.assert_set_equal(a: Set, b: Set, message: str | None = None) → None
```

Assert that `a` equals `b` and that both are of type `set`.

Parameters:

- **a** – a set
- **b** – a set
- **message** – the message to prepend to the error message

Raises:
AssertionError – if `a` and `b` are not equal or not sets

```
htf.assertions.assert_str_equal(a: str, b: str, message: str | None = None) → None
```

Assert that `a` equals `b` and that both are of type `str`.

Parameters:

- **a** – a string
- **b** – a string
- **message** – the message to prepend to the error message

Raises:
AssertionError – if `a` and `b` are not equal or not strings

```
htf.assertions.assert_true(expression: bool, message: str | None = None) → None
```

Assert that `expression` is `True`.

Parameters:

- **expression** – the boolean expression to be validated
- **message** – the message to prepend to the error message

Raises:
AssertionError – if `expression` is not `True`

```
htf.assertions.assert_tuple_equal(a: Tuple, b: Tuple, message: str | None = None) → None
```

Assert that `a` equals `b` and that both are of type `tuple`.

Parameters:

- **a** – a list
- **b** – a list
- **message** – the message to prepend to the error message

Raises:

AssertionError – if `a` and `b` are not equal or not tuples

```
htf.assertions.get_equality_asserter(type_: Type) → Callable[[...], None]
```

Return the registered equality asserter for `type_`.

Parameters:
type – a type
Returns:
the registered equality asserter for `type`
Raises:
KeyError – if `type_` has no registered equality asserter

```
htf.assertions.set_diff_threshold(threshold: int) → None
```

Set the diff threshold to `threshold`:

Parameters:
threshold – the maximum number of characters to use difflib

```
htf.assertions.set_equality_asserter(type_: Type, asserter: Callable[[...], None]) → None
```

Register a new type equality asserter used in `assert_equal` and `assert_not_equal`.

Parameters:

- **type** – the type the checker is used for
- **asserter** – callable equality asserter object that supports parameters `a`, `b` and `message`

```
htf.assertions.set_repr_length(length: int) → None
```

Set the maximum representation length to `length`

Parameters:
length – the maximum representation length

For legacy tests the `htf.TestCase` offers wrappers for all built-in assertions and also non-pep8 wrappers to support legacy tests.

Behavior Driven Development

Behavior Driven Development (BDD) is an agile software development technique. BDD enables non-technical participants to participate in writing tests.

The HILSTER Testing Framework enables users to write features in a gherkin-like language with add-ons that can be run as tests with seamless integration into the existing ecosystem.

Tests are written in form of Features, Rules, Scenarios and Steps, which are either Given, When, or Then steps. It is also possible to create Data Driven Feature Tests.

Thus every step is implemented in Python based on HILSTER Testing Framework.

HILSTER Testing Framework also offers requirements coverage in combination with the Dashboard.

Features

A feature file describes a feature in natural language with scenarios describing actions and the expected outcome. The filename ends with `.feature` with `UTF-8 encoding`. Every feature file must contain one single feature. Every feature can contain one or multiple scenarios or scenario outlines.

A feature may contain further keywords to describe how tests may run.

Lines starting with `#` are comments.

Scenarios

Scenarios are “tests” which consist of `Given`, `When`, and `Then` statements. They describe preconditions, actions as well as the expected outcomes.

Steps

`Given` steps specify preconditions, which describe the state of the environment. They are used to put the tested system into an initial state.

`When` steps specify conditions, for example `when car turns right` is a condition. They are used to take actions.

`Then` steps specify assertions and serve for checking if conditions are met. They are used to observe outcomes.

Each statement can be repeated with `And` as well as `But`.

HILSTER Testing Framework finds the correct implementation of these steps so that they can be executed as a test.

Writing Features

A feature is written in a file named `<feature-name>.feature`.

The feature description is written with the `Feature:` keyword followed by a feature name and an optional indented documentation string in the next lines.

```
Feature: Example features without info string
```

```
Feature: Example features with info string
  This documentation string is optional.
```

Writing Scenarios

A scenario is described inside a `Scenario:` keyword. It is indented compared to the feature.

```
Feature: Heating
  Scenario: Heat up
    Given the refrigeration machine controller is running at 100 rpm
    And the environmental temperature is 21 °C
    When the initial temperature is -100 °C
    Then the refrigeration machine controller does not cool down
```

Example Feature

The following example contains a very simple feature with a single scenario.

```
Feature: This is an example feature
    This features is only used as an example

    Scenario: Example
        Given we write an example
        When the interested reader reads it
        Then he will understand how BDD works
```

Complex Scenarios

As an extension to BDD, scenarios support repeated keywords which allow you to create more complex scenarios.

```
Feature: This is an example feature
    This features is only used as an example

    Scenario: Complex Example
        Given we write an example
        When the interested reader reads it
        Then he will understand how BDD works

        Given we write another example
        When the interested reader reads it, too
        Then he will understand how BDD works for sure
```

Implementing Steps

Steps are implemented in Python as methods that are decorated with `htf.given`, `htf.when` and `htf.then` decorators.

The following code shows the implementation of the corresponding steps so that this feature can be run as a test.

```
@htf.given("we write an example")
def we_write_an_example():
    pass

@htf.when("the interested reader reads it")
def the_interested_reader_reads_it():
    pass

@htf.then("he will understand how BDD works")
def he_will_understand_how_bdd_works():
    pass
```

The corresponding steps are found via pattern matching. Details are described later.

Steps can be async, too.

```
@htf.given("we write an example")
async def we_write_an_example():
    pass

@htf.when("the interested reader reads it")
async def the_interested_reader_reads_it():
    pass

@htf.then("he will understand how BDD works")
async def he_will_understand_how_bdd_works():
    pass
```

```
htf.given(precondition: str, parser: str | None = None, extra_types: Dict[Any, Any] | None =
None, converters: Dict[str, Any] | None = None) → Callable[[Callable[[...], Any]], Any]
```

Decorates a method to make it an implementation of a given step in a behavior driven test. See also https://cucumber.io/docs/gherkin/reference/#given .

Parameters:

- **precondition** – gherkin feature precondition
- **parser=None** – the parser to parse the parameters from the precondition
- **extra_types=None** – extra types for parsing
- **converters=None** – optional converters that are applied to parameters after parsing

```
htf.when(condition: str, parser: str | None = None,
extra_types: Dict[Any, Any] | None = None, converters: Dict[str, Any] | None = None) →
Callable[[Callable[[...], Any]], Any]
```

Decorates a method to make it an implementation of a when step in a behavior driven test. See also https://cucumber.io/docs/gherkin/reference/#when .

Parameters:

- **condition** – gherkin feature condition
- **parser=None** – the parser to parse the parameters from the condition
- **extra_types=None** – extra types for parsing
- **converters=None** – optional converters that are applied to parameters after parsing

```
htf.then(assertion: str, parser: str | None = None,
extra_types: Dict[Any, Any] | None = None, converters: Dict[str, Any] | None = None) →
Callable[[Callable[[...], Any]], Any]
```

Decorates a method to make it an implementation of a then step in a behavior driven test. See also https://cucumber.io/docs/gherkin/reference/#then .

Parameters:

- **assertion** – gherkin feature assertion
- **parser=None** – the parser to parse the parameters from the assertion
- **extra_types=None** – extra types for conversion
- **converters=None** – optional converters that are applied to parameters after parsing

Running Features

Run `htf` by giving the specifier for `feature` file, and also the `<step-implementation>.py` file. Features and normal tests can also be mixed up.

In Python you can run:

```
htf.main(tests=["steps.py", "example.feature"])
```

And on the command-line you can run:

```
htf steps.py example.feature
```

Steps with Parameters

You can implement steps with parameters, too. This allows to use one step implementation for more than one step in a feature file.

```
Feature: Feature with parameters in steps

    Scenario: Monkey eats many Bananas
        Given the monkey is awake
        When the monkey eats 10 bananas
        Then he is not hungry anymore

    Scenario: Monkey eats two Bananas
        Given the monkey is awake
        When the monkey eats 2 bananas
        Then he is still hungry
```

The `When` step with the parameter to read the number of bananas is implemented in the following example.

```
@htf.when("the monkey eats {number_of_bananas} bananas")
def monkey_eats_bananas(number_of_bananas):
    print(f"Monkey eats {number_of_bananas} bananas")
```

This also works for the other steps.

There are also other parsers for more complex situations.

Long Lines

Long lines can be split up to multiple lines by adding a `\` at the end.

```
Feature: Feature with long lines

    Scenario: Monkey eats many Bananas
        Given the monkey is awake
        When the monkey eats \
             10 bananas
        Then he is not \
             hungry \
             anymore
```

Multi-Language Support

Features can be written in natural languages and do not need to use the English language. The first line of a feature file controls the language. The default language is English (`en`).

The following example uses German as the language.

```
# language: de
Funktion: Heizen
  Szenario: Aufheizen
    Angenommen die Maschine läuft mit 100 Umdrehungen pro Minute
```

```
    Wenn die initiale Temperatur 100 °C beträgt
    Dann erreicht die Temperatur der Maschine nicht 120 °C
```

Context Fixture

If you need to pass data between steps you can use the `htf.fixtures.context` fixture.

Other fixtures can be used like expected. Step parameters are supplied on the left side.

```
@htf.when("the monkey eats {number_of_bananas} bananas")
def monkey_eats_bananas(number_of_bananas, context):
    print(f"Monkey eats {number_of_bananas} bananas")
    context.number_of_bananas = int(number_of_bananas)

@htf.then("he is not hungry anymore")
def monkey_is_not_hungry_anymore(context, assertions):
    assertion.assert_greater(context.number_of_bananas, 5, "The monkey did not eat enough bananas!")

@htf.then("he is still hungry")
def monkey_is_still_hungry(context, assertions):
    assertion.assert_less_equal(context.number_of_bananas, 5,
                                "The monkey ate too much bananas! The monkey is dead now.")
```

To set the actual result of a step, the fixture `htf.fixtures.context` contains `current_step`, which is the `htf.fixtures.step` instance in the current context.

```
@htf.then("the current number of bananas is {number_of_bananas}")
def check_current_number_of_bananas(number_of_bananas, context):
    current_number_of_bananas = 10
    context.current_step.set_actual_result(current_number_of_bananas)
    htf.assert_almost_equal(current_number_of_bananas, number_of_bananas)
```

Attach Data to Steps

You can attach more complex data to steps compared to parameters by adding a data table or text in the feature. The data is passed via the `htf.fixtures.context` fixture as `table` or `text` attribute.

```
Feature: Feature with Data

    Scenario: Monkey feeding
        When the monkey eats bananas
            | i | name   |
            | 1 | first  |
            | 2 | second |
        And the monkey eats bananas and says
            """
            This text is said while eating bananas
            """
            | i | name   |
            | 3 | third  |
```

```
            | 4 | fourth |
        Then the monkey says
            """
            Baarrrb.
            """
```

The implementation of the steps looks like the following

```
@htf.when("the monkey eats bananas")
def monkey_eats_bananas(context):
    for item in context.table:
        print(f"Eat the {item['i']}th banana called {item['name']}")

@htf.when("the monkey eats bananas and says")
def monkey_eats_bananas(context):
    for item in context.table:
        print(f"Eat the {item['i']}th banana called {item['name']}")
        print(f"Monkey: {context.text}")

@htf.then("the monkey says")
def monkey_says(context):
    print(f"Monkey: {context.text}")
```

Tagging

HILSTER Testing Framework offers the possibility to add textual tags to features and scenarios as well as scenario outlines.

To do this use the `Tags:` keyword followed by a list of comma-separated strings.

```
Feature: Feature with Tags

    Tags: Tag-1, Tag-2, Tag-3

    Scenario: Scenario with Tags
        Tags: Tag-4, Tag-5, Tag-6
        # ...
```

Tags are combined and put into the test result and thus can be used to filter text by tags as mentioned in Tagging.

Requirements

HILSTER Testing Framework offers the possibility to add textual requirements to features and scenarios as well as scenario outlines.

To do this use the `Requirements:` keyword followed by a list of comma-separated strings.

```
Feature: Feature with Requirements

    Requirements: Req-1, Req-2, Req-3

    Scenario: Scenario with Requirements
        Requirements: Req-4, Req-5, Req-6
        # ...
```

The requirements are combined and put into the test result and thus can be used to perform requirements coverage in combination with the Dashboard.

Implemented Tests

HILSTER Testing Framework offers the possibility to add links to implemented tests to features and scenarios.

To do this use the `Implements:` keyword followed by a list of comma-separated strings. Each entry can be a url or a simple id.

```
Feature: Feature with implemented tests

    Implements: Test-1, https://host:port/path/to/Test-2

    Scenario: Scenario with implemented tests
        Implements: Test-3
        # ...
```

The implemented tests are added to the test report and used to send test results back to ALM/QMS systems.

Background Steps

The keyword `Background:` can be used when scenarios inside a feature have intersecting preconditions. It contains only `Given` preconditions that are automatically run in every scenario before running the other steps. It can be used to realize common preconditions.

```
Feature: Refrigeration machine can preserve internal heat correctly

    Background:
        Given the refrigeration machine controller is running at 100 rpm

    Scenario: Heat up
        Given the environmental temperature is 21 °C
        # ...

    Scenario: Cool down
        Given the environmental temperature is 21 °C
        # ...
```

Preconditions for `Background:` are implemented like normal given steps with `htf.given`.

Rules

The `Rule:` keyword is optionally used for grouping together related scenarios when they belong to the same feature.

```
Feature: Monkey movements are correct
  Rule: Monkeys can run if they are healthy
    Scenario: Baboon can run
        Given baboon is healthy and adult
        Then baboon can run
    Scenario: Chimp
        Given chimp is healthy and adult
        Then chimp can run

  Rule: Another rule
     # ...
```

Scenario Outlines

A scenario outline is a scenario template that dynamically generates scenarios. It is the behavior driven way to realize Data Driven Testing. It consists of steps like a normal scenario that has template variables in it and one data source called `Examples`. When the scenarios are generated the template variables are replaced automatically.

```
Feature: Feature with Scenario Outline
  Scenario Outline: Machine does not cool down
    Given the refrigeration machine controller is running at <rpm> rpm
    When the initial temperature is <temperature> °C
    Then the refrigeration machine controller does not cool down

    Examples:
       | rpm | temperature |
       |  78 |         -10 |
       |  80 |          21 |
       |  90 |          25 |
       | 100 |          38 |
```

Running the examples leads to multiple dynamically generated scenarios with parameters:

```
Scenario: Machine does not cool down [rpm=78, temperature=-10] (Feature: ...)
Scenario: Machine does not cool down [rpm=80, temperature=21] (Feature: ...)
Scenario: Machine does not cool down [rpm=90, temperature=25] (Feature: ...)
Scenario: Machine does not cool down [rpm=100, temperature=38] (Feature: ...)
```

The implementation can be done with normal steps with parameters for example.

```
htf.given("the refrigeration machine controller is running at <rpm> rpm")
def machine_running_at(rpm):
    # this is called once for every line in the data table (78, 80, 90 and 100)

htf.when("the initial temperature is {temperature} °C)
def initial_temperature(temperature):
    # this is called once for every line in the data table (-10, 21, 25 and 38)
```

Scenario Outline Data Sources

Scenario outlines support multiple different data sources which can be data-tables, external CSV-, JSON, YAML-files and also Python generators. Every data source can have an optional title.

Data-Tables Data Sources

Data-tables are textual tables supplied directly in the feature file. The first line is a header. All other lines are data lines. A data table is parsed into a list of dictionaries with the keys from the header line.

```
Feature: Feature with Data-Table
    Scenario Outline: Data-Table Scenario
        Given there is a data-table with <key> and <value>

    Examples: Data-Table
        |  key | value |
        |  foo |   bar |
        | ding |  dong |
```

Running the examples leads to multiple dynamically generated scenarios with parameters:

```
Scenario: Data-Table Scenario [key=foo, value=bar] (Feature: ...)
Scenario: Data-Table Scenario [key=ding, value=dong] (Feature: ...)
```

CSV Data Sources

Instead of using data-tables you can also supply external data via a CSV file.

To do this you can supply an example with the `CSV:` keyword followed by a CSV filename. The filename can be an absolute path or a relative path. Relative paths are relative to the feature file where they appear.

The following example show the contents of `data.csv`

```
key,value
foo,bar
ding,dong
```

The following feature shows how to use CSV data.

```
Feature: External CSV Data
    Scenario Outline: CSV Scenario Outline
        Given there is a csv-data-file with <key> and <value>

    Examples: CSV Data-Source
        CSV: data.csv
```

Running the examples leads to multiple dynamically generated scenarios with parameters:

```
Scenario: CSV Scenario Outline [key=foo, value=bar] (Feature: ...)
Scenario: CSV Scenario Outline [key=ding, value=dong] (Feature: ...)
```

JSON Data Sources

JSON data sources are also supported via the `JSON:` keyword followed by a JSON filename.

The filename can be an absolute path or a relative path. Relative paths are relative to the feature file where they appear.

The following example show the contents of `data.json`

```
[
    {"key": "foo", "value": "bar"},
    {"key": "ding", "value": "dong"}
]
```

The contents have to be a list of dictionaries.

The following feature shows how to use CSV data.

```
Feature: External JSON Data
    Scenario Outline: JSON Scenario Outline
        Given there is a json-data-file with <key> and <value>

    Examples: JSON Data-Source
        JSON: data.json
```

Running the examples leads to multiple dynamically generated scenarios with parameters:

```
Scenario: JSON Scenario Outline [key=foo, value=bar] (Feature: ...)
Scenario: JSON Scenario Outline [key=ding, value=dong] (Feature: ...)
```

YAML Data Sources

YAML data sources are also supported via the `YAML:` keyword followed by a YAML filename.

The filename can be an absolute path or a relative path. Relative paths are relative to the feature file where they appear.

The following example show the contents of `data.yml`

```
- key: foo
  value: bar
- key: ding
  value: dong
```

The contents have to be a list of dictionaries.

The following feature shows how to use CSV data.

```
Feature: External YAML Data
    Scenario Outline: YAML Scenario Outline
        Given there is a yaml-data-file with <key> and <value>

    Examples: YAML Data-Source
        YAML: data.yml
```

Running the examples leads to multiple dynamically generated scenarios with parameters:

```
Scenario: YAML Scenario Outline [key=foo, value=bar] (Feature: ...)
Scenario: YAML Scenario Outline [key=ding, value=dong] (Feature: ...)
```

Python Generators Data Sources

Python generators can also be used as data sources via the `Generator:` keyword followed by a generator title.

Every data generator needs to be decorated with `htf.data_generator`. The following Python code can be used as data sources.

```
@htf.data_generator("Python-based Generator")
def data_generator() -> Generator[Dict[str, str], None, None]:
    yield dict(key="foo", value="bar")  # this is the most efficient way
    yield dict(key="ding", value="dong")
    # ...

@htf.data_generator("Python-based Data Source with Lists")
def data_source_with_list() -> List[Dict[str, str]]:
    return [  # this also works
        dict(key="foo", value="bar"),
```

```
        dict(key="ding", value="dong")
        # ...
    ]
```

Async Python generators are supported, too.

```
@htf.data_generator("Python-based Generator")
async def data_generator() -> AsyncGenerator[Dict[str, str], None]:
    yield dict(key="foo", value="bar")  # this is the most efficient way
    yield dict(key="ding", value="dong")
    # ...
```

The data generator must return an iterable so you can make it a Python generator or let it return an iterable object like a list for example.

The following feature shows how to use a Python generator as a data source.

```
Feature: Python Generator Data Source
    Scenario Outline: Data Source Scenario Outline
        Given there is a data-source yielding <key> and <value>

    Examples: Python Generator Data-Source
        Generator: Python-based Generator
```

Running the examples leads to multiple dynamically generated scenarios with parameters:

```
Scenario: Data Source Scenario Outline [key=foo, value=bar] (Feature: ...)
Scenario: Data Source Scenario Outline [key=ding, value=dong] (Feature: ...)
```

```
htf.data_generator(pattern: str, parser: str | None = None) → Callable[[Callable[[...],
Any]], Any]
```

Decorates a method to make it an implementation of a data generator used for examples in a behavior driven test.

Parameters:

- **pattern** – the pattern to match the title
- **parser=None** – the parser to match the name of the example

Multiple and Combined Data Sources

Every scenario outline can have multiple data sources that can also be mixed.

```
Feature: Mixed External Data
    Scenario Outline: Scenario Outline with Mixed Data
```

```
        Given there is a data-table with <key> and <value>

    Examples:
        |  key | value |
        |  foo |   bar |
        # CSV: data.csv
        # JSON: data.json
        # YAML: data.yml
        # Generator: Python-based Generator
        # etc.
```

Transforming Data Sources

Sometimes you need to precalculate values based on data-sources. To do this HILSTER Testing Framework supports data-transformation.

A data transformation is created by decorating a method with the `htf.transformation` decorator. The method is called for every line of the data source. To use a data-transformation the data source (`Example:`) need to have a name supplied.

In the following example the Addition functionality for a calculator is tested. The data-source only contains to addends but not the required result. To check if the calculator is working properly the result needs to be precalculated.

```
Feature: Addition

  Scenario Outline: Addition
    Given the calculator is cleared
    When <a> is added to <b>
    And the equals button is pressed
    Then the result is <result>

    Examples: Addition
      |   a |   b |
      |   1 |   1 |
      |   2 |  12 |
```

To implement a data-transformation hook you could use the following code:

```
@htf.transformation("Addition")
def expected_addition_result(data):
    data["result"] = int(data["a"]) + int(data["b"])
    return data
```

Running the examples leads to multiple dynamically generated scenarios with parameters where `result` is generated on the fly.

Transformations can be async, too.

```
@htf.transformation("Addition")
async def expected_addition_result(data):
    data["result"] = int(data["a"]) + int(data["b"])
    return data
```

```
Scenario: Addition [a=1, b=2, result=3] (Feature: ...)
Scenario: Addition [a=2, b=12, result=14] (Feature: ...)
```

```
htf.transformation(pattern: str, parser: str | None = None) → Callable[[Callable[[...],
Any]], Any]
```

Decorates a method to make it an implementation of a data transformation hook used for examples in a behavior driven test.

Parameters:

- **pattern** – gherkin examples pattern
- **parser=None** – the parser to match the name of the example

Skipping Features, Scenarios or Scenario Outlines

You can skip features, scenarios or scenario outlines by adding a line `Skip: <reason>` to your feature.

To skip a whole feature you can use:

```
Feature: Skipped Feature
    Skip: this feature is skipped
    Scenario: Skipped Feature Scenario
        Given there is a skipped feature
```

To skip a scenario you can use:

```
Feature: Skipped Feature
    Scenario: Skipped Feature Scenario
        Skip: this scenario is skipped
        Given there is a skipped feature
```

To skip a scenario outline you can use:

```
Feature: Skipped Feature
    Scenario Outline: Skipped Feature Scenario Outline
        Skip: this scenario is skipped
        Given there is a skipped feature

        Examples:
```

```
        |  key | value |
        |  foo |   bar |
```

Setup and Tear Down Code

It is possible to create setup and tear down code for features and scenarios, too.

To do this you can use `htf.feature` and `htf.scenario`.

The decorated methods can be returning methods to only implement setup code or can be implemented as generators yielding one element to implement setup and tear down code.

Fixtures and parameters are added like for the other step implementations.

```
Feature: Feature with Setup and Tear Down code

    Scenario: Setup and Tear Down
        Given the setup was run
        when actions are performed
        then an outcome will be available
```

In the following example only setup code for a feature and a scenario is implemented.

```
@htf.feature("Feature with Setup and Tear Down code")
def feature_set_up(context):
    # your feature setup code comes here

@htf.scenario("Setup and Tear Down")
def scenario_set_up(context):
    # your scenario setup code comes here
```

And in the following example setup and tear down code is implemented.

```
@htf.feature("Feature with Setup and Tear Down code")
def feature_set_up_and_tear_down(context):
    # your feature setup code comes here
    yield  # now the scenario is run
    # your feature tear down code comes here

@htf.scenario("Setup and Tear Down")
def scenario_set_up(context):
    # your scenario setup code comes here
    yield  # now the scenario is run
    # your scenario tear down code comes here
```

Setup and tear down code can be async, too.

```
htf.feature(pattern: str, parser: str | None = None, extra_types: Dict[Any, Any] | None =
None, converters: Dict[str, Any] | None = None) → Callable[[Callable[[...], Any]], Any]
```

Decorates a method to make it an implementation of a given feature in a behavior driven test.

It can be implemented as a normal method with a return value to implement set up code or as a generator to realize set up and tear down code for a feature.

Parameters:

- **pattern** – gherkin feature pattern
- **parser=None** – the parser to parse the parameters from the precondition
- **extra_types=None** – extra types for parsing
- **converters=None** – optional converters that are applied to parameter after parsing

```
htf.scenario(pattern: str, parser: str | None = None, extra_types: Dict[Any, Any] | None =
None, converters: Dict[str, Any] | None = None) → Callable[[Callable[[...], Any]], Any]
```

Decorates a method to make it an implementation of a given scenario in a behavior driven test.

It can be implemented as a normal method with a return value to implement set up code or as a generator to realize set up and tear down code for a scenario.

Parameters:

- **pattern** – gherkin feature pattern
- **parser=None** – the parser to parse the parameters from the precondition
- **extra_types=None** – extra types for parsing
- **converters=None** – optional converters that are applied to parameters after parsing

Parsing Steps

HILSTER Testing Framework offers three different parsers to parse steps and to find the implementation for a step by its name.

The default parser is based on the `parse` module. There is an extended version of `parse` called `cfparse`. And there is also a regular-expression parser (`re`).

All parsers support pattern matching and type casting.

How Parsing and Matching Works

HILSTER Testing Framework collects all implemented steps and features. When features, scenarios and steps are run the implementing step is looked up by iterating over all collected steps with the corresponding scope and matching the current step to the pattern using a parser.

If there is one match the step will be executed with optional parameters extracted. Parameters can also be converted to other types before calling the implementing method.

Using the Default Parser (parse)

The default step parser is based on the `parse` packet that is the opposite of Python's `format` method. It supports the Format String Syntax.

Parameters are put between `{` and `}` (e.g. `{number}`) and can have type casts in the format expression in the form of `{[parameter name]:[format specification]}`.

The parsed parameters are passed to the implementing method by its name.

The following Gherkin example describes addition.

```
Feature: Addition

  Scenario: Addition
    Given the calculator is cleared
    When 1 is added to 2
    And the equals button is pressed
    Then the result is <result>
```

The following example parses `a` and `b` that are added as strings that are added.

```
@htf.when("{a} is added to {b}")
def add(a, b, context):
    context.expected_result = int(a) + int(b)  # this type-cast has to be performed here!
```

In the example the parameters `a` and `b` are passed as strings so they need to be converted to `int` in the implementation.

To automatically convert `a` and `b` to int while parsing you can change the format string to:

```
@htf.when("{a:g} is added to {b:g}")
def add(a, b, context):
    context.expected_result = a + b  # a and b are int
```

You can find more information about the type conversion on GitHub parse repository.

Using the Regular-Expression-Parser (re, regex)

The `re` parser uses regular-expressions to do the parsing. Thus it is the most flexible parser but is more complex.

To enable the CF-Parser the `parser` argument of the step-decorator has to be set with `cf` or `cfparse` and `converters` can to be supplied optionally.

`converters` is a dictionary of callables where the keys are the names of the parameters of the implementing method.

The regular-expression must make use of named subgroups (e.g. `r"(?P<parameter name>\w+)"`). You can find more information about named subgroups in the Python re Module.

```
Feature: Addition

  Scenario: Addition
    Given the calculator is cleared
    When 1 is added to 2
    And the equals button is pressed
    Then the result is <result>
```

The following example parses `a` and `b` that are added as strings that are added.

```
@htf.when(r"(?P<a>\d+) is added to (?P<b>\d+)", parser="re")
def add(a, b, context):
    context.expected_result = int(a) + int(b)  # this type-cast has to be performed here!
```

In the example the parameters `a` and `b` are passed as strings so they need to be converted to `int` in the implementation.

To automatically convert `a` and `b` to int while parsing you can change the format string to:

```
@htf.when(r"(?P<a>\d+) is added to (?P<b>\d+)", parser="re", converters=dict(a=int, b=int))
def add(a, b, context):
    context.expected_result = a + b  # a and b are int
```

Using the CF-Parser (cfparse, cf)

The CF-Parser extends the default with support for cardinality fields, and it automatically creates type variants to parse repeated parameters.

To enable the CF-Parser the `parser` argument of the step-decorator has to be set with `cf` or `cfparse` and `extra_types` have to be supplied, too.

The following examples show how to accumulate a list of integers with a variable length.

The following example accumulates a list of numbers with variable amount.

```
Feature: Accumulation
  Scenario: Accumulate 1
    When 1 is accumulated
    Then the result is 1

  Scenario: Accumulate 2
    When 1, 2 is accumulated
    Then the result is 3

  Scenario: Accumulate many
```

```
    When 1, 2, 3, 4, 5, 6, 7, 8, 9 is accumulated
    Then the result is 45
```

The Python-implementation looks like

```
import parse

@parse.with_pattern(r"\d+")
def parse_number(text):
     return int(text)

extra_types = dict(Number=parse_number)

@htf.when("{numbers:Number+} is accumulated", parser="cf", extra_types=extra_types)
def accumulate(numbers, context):
    context.accumulated_numbers = sum(numbers)

@htf.then("the result is {result:g}")
def the_result_is(result, context):
    htf.assert_equal(result, context.accumulated_numbers)
```

You can find more information about cardinality field support on GitHub parse-type repository.

Getting Available Steps and Statements

As a developer you need to get all available steps and statements without looking into the code all the time.

This can be done using `htf.get_statements` and `htf statements <steps/test specifiers>`.

On the command-line you can run:

```
htf statements steps.py

Collecting statements

Given statements:
Given the refrigeration machine controller is running at {speed:g} rpm

When conditions:
When the initial temperature is {temparature:g} °C
When {numbers:Number+} is accumulated

Then assertions:
Then the result is {result:g}
Then the refrigeration machine controller is cooling down
Then the refrigeration machine controller does not cool down

Features:
.*

Scenarios:
.*
```

```
Transformations:
X and Y data

Data Generators:
Data Generator
```

And from Python you can use `htf.get_statements`:

```
htf.get_statements("steps.py")
>>> {
    'given': [
        'Given the refrigeration machine controller is running at {speed:g} rpm'
    ],
    'when': [
        'When the initial temperature is {temparature:g} °C',
        'When {numbers:Number+} is accumulated'
    ],
    'then': [
        'Then the refrigeration machine controller is cooling down',
        'Then the refrigeration machine controller does not cool down',
        'Then the result is {result:g}'
    ],
    'features': ['.*'],
    'scenarios': ['.*'],
    'transformations': ['X and Y data'],
    'data_generators': ['Data Generator']
}
```

```
htf.get_statements(tests: Any | List[Any] = '__main__') → Dict[str, List[str]]
```

Get a dict of behavior driven development statements found in the specified tests.

Parameters:
tests=None – a test-specifier or a list of test-specifiers (folder, file, test-case, test-case-method, module)
Returns:
a dict with all behavior driven development statements found
Return type:
dict

Examples

You can find examples in the htf-demo, too.

htf — Command line utility

The HILSTER Testing Framework comes with a command line utility called `htf`.

`htf` can be used to run tests, open docs and view the included open-source licenses.

Running tests

To run all tests within the current folder simply run

```
htf
```

By default `htf` creates an HTML-testreport called `testreport.html` in the current folder.

Specifying tests

`htf` may be followed by `run` and `<tests>`. You can also omit `run` and only supply test-specifiers, eg.

```
htf run <tests>
```

respectively

```
htf <tests>
```

`<tests>` is a list of test-specifiers. They are separated by whitespaces.

A test-specifier may be a python module or a feature file, a globbing-expression for python modules (e.g. `test_*.py`), a globbing expression for feature files (e.g. `features/*.feature`), a folder or an import string. Test-specifiers can be mixed.

If a test-specifier is a python module the file is imported and all testable objects are collected into a test suite that is used for the test run.

```
htf test_1.py
```

If a test-specifier ist a globbing-expression the expression is evaluated and all matching files are imported.

```
htf test_\*.py
```

Note: the `\*` lets not `Bash` (or any other shell) evaluate it and passes it to `htf`.

If a test-specifier is a folder the folder is searched recursively and all python modules found within (see File pattern) are imported and all testable objects are collected into a test suite. Subsequently the test suite is executed.

Consider the folder `tests`, which contains python modules with unit tests. To execute all test cases in the folder `tests`, simply run

```
htf tests
```

In case `tests` is an importable package or module, located in the python search path, the names might collide. In this case, simply use

```
htf tests/
```

If a test-specifier is an import string, this may be a package, a module, a class (inheriting from `htf.TestCase`) or a method.

If a test-specifier is a package or a module, it is imported and containing tests are executed.

```
htf tests  # Example for a python package containing tests
htf tests.test_example  # Example for a python module containing tests
```

If a class is used it should be a testcase inheriting from `htf.TestCase`. This testcase is used in a test suite and all tests are run.

```
htf tests.test_example.ExampleTestCase # Example for a class containing tests
```

If a method is used it should be a method within a test case.

```
htf tests.test_example.ExampleTestCase.test_one # Example for a test method
```

Running multiple tests

To run multiple tests within a test run simply add more test-specifiers at the end.

```
htf test1 test2 test3
```

Different test-specifiers can also be mixed

```
htf tests/ test1.Test1TestCase test2.py
```

Test-title

To specify the test's title add the `-T` or `--title` option followed by a string that will be used as the title. If the title contains whitespaces, make sure to put quotes around it.

```
htf --title="This is my first test" tests/
```

HTML-report

By default `htf` generates an HTML-report called `testreport.html`. To specify an HTML-report add `-H` or `--html-report`.

```
htf --html-report="tests.html" tests/
```

This option may be repeated to generate multiple HTML-reports.

```
htf --html-report="tests1.html" --html-report="tests2.html" tests/
```

To open all HTML-reports use `-o` or `--open-report`.

```
htf --html-report="tests.html" -o tests/
```

JUnit-XML-report

`htf` is also able to generate JUnit-XML-compatible XML-reports. The reports can be evaluated by different tools, eg. Jenkins.

To specify an XML-report add `-x` or `--junit-xml-report`.

```
htf --junit-xml-report=results.xml tests/
```

This option may also be repeated.

JSON-report

`htf` is also able to generate JSON-reports.

To specify a JSON-report add `-j` or `--json-report`.

```
htf --json-report=results.json tests/
```

This option may also be repeated.

YAML-report

`htf` is also able to generate YAML-reports.

To specify a YAML-report add `-Y` or `--yaml-report`.

```
htf --yaml-report=results.yaml tests/
```

This option may also be repeated.

Report Server

To send test results to a test report server (hre) use `-r` or `--report-server`.

```
htf -r https://reports.host/project/token
htf --report-server=https://reports.host/project/token
```

To disable SSL-verfification you can set the environment variable `HTF_INSECURE` to any value.

```
HTF_INSECURE=1 htf -r https://reports.host/project/token  # Linux
set HTF_INSECURE=1  # Windows
htf -r https://reports.host/project/token  # Windows
```

Draft Reports

To create draft reports after each test use `-1` or `--create-drafts`.

```
htf -1
```

or

```
htf --create-drafts
```

Minimized Report Size

To minimize the report size use the `-M` or `--minimize-report-size` switch. When used for example captured stdout and stderr is only kept for failed tests. By default the report size is not minimized.

```
htf -M tests/
```

Shuffle Tests

To shuffle tests before running add `-S` or ``--shuffle`.

```
htf --shuffle tests/
```

To set the shuffle seed use `--shuffle-seed=<int>|last`. If `'last'` is used the seed is loaded from `'shuffle_seed.txt'`. If the file cannot be found one seed is created.

After a seed is created it is stored to be used next time. Using the seed the shuffle result can be reproduced.

```
htf --shuffle --shuffle-seed=12345 tests/
htf --shuffle --shuffle-seed=last tests/
```

Filename-templates

Filenames for reports can include prepopulated templates.

Available templates:

- `{{title}}` - the title as a filename
- `{{date}}` - the current date `"%Y-%m-%d_%H-%M-%S"`
- `{{hostname}}`, `{{host}}` or `{{node}}` - the hostname
- `{{htf_version}}` or `{{version}}` - the htf version, eg. `htf_1.0.0`
- `{{python_version}}` - the python version, eg. `Python_3.6.1`

To generate an HTML-report containing the current node and the current date use

```
htf -H test_{{host}}_{{date}}.html tests/
```

Fail fast

If `-f` or `--fail-fast` is used the test run ends after the first failing test.

File pattern

If a test-specifier is a folder the default pattern used for file discovery is `test*.py`. The pattern can be changed using the `-p` or `--pattern` options.

```
htf -p \*.py tests/
```

Wildcards (*) are supported. Depending on your shell, escaping must be applied.

Tagging

To select tests using a logical expression use the `-t` or `--tags` option followed by an expression.

```
htf --tags='foo|bar'
```

To list all available tags use the `tags` commands.

```
htf tags /path/to/test
```

To use tags for fixtures use `-F``or ``--fixture-tags` in addition.

```
htf -F hardware
htf --fixture-tags=hardware
```

Metadata

To add metadata about the test run use the `-m` or `--metadata` option followed by a comma-separated list of key-value pairs.

These options can also be repeated.

This feature can for example be used to add information about a tested firmware version, etc.

```
htf -m key1=value1,key2=value2 --metadata key3=value3  # etc.
```

The metadata will occur in the test reports.

Overwriting Settings

You can overwrite settings and even hierarchical settings from command line using `-s <name>=<value>` or `--setting <name>=<value>`.

Hierarchies in settings can be realized with a `.` e.g. `-s top.center.bottom=1`.

For example the following call

```
htf -s a=1,b=2 -s top.sub=3
```

leads to settings looking like

```
{
    'a': '1',
    'b': '2',
    'top': {
        'sub': '3'
    }
}
```

Please note that all supplied settings are strings and you have to care about the casting when these will be used.

This option support to do manual experiments, etc.

Parameters

You can add parameters to test runs from command line to control the behavior of tests and fixtures. Add `-P <name>=<value>` or `--parameter <name>=<value>` to the call.

The parameters are available via the `htf.fixtures.Parameters` fixture.

```
htf -P name1=value1,name2=value2 --parameter=name3=value3
```

The parameters will also appear in the test result metadata.

Bash-completion

`htf` supports bash-completion which provides automatic command completion for the `bash` shell.

Bash-completion completes the first positional commands (`run`, `docs` and `licenses`, etc.) but also expands test-specifiers (eg. file names, packages, modules, classes and methods) and other parameters like `--html-report`, `--metadata`, etc.

Using the bash-completion makes it easy to specify tests (without typing-errors).

Let's assume you a have an importable python package called `tests`. Two test-modules called `test_1.py` and `test_2.py` are located within `tests`.

Both contain a test-case called `TestCase1` and `TestCase2`, respectively.

To easily run `tests.test_1.TestCase1`, type

```
htf te<TAB>
    tests    tests.
```

```
htf tests.<TAB>
htf tests.test_<TAB>
    tests.test_1    tests.test_1.    tests.test_2    tests.test_2.
htf tests.test_1.<TAB>
htf tests.test_1.TestCa<TAB>
htf tests.test_1.TestCase<TAB>
    tests.test_1.TestCase    tests.test_1.TestCase.
htf tests.test_1.TestCase<ENTER>
# ...
```

Enable Bash-completion

To enable bash-completion type

```
eval $(htf complete)
```

To persistently enable bash completion, add the command to your `~.bashrc`. alternatively put it into a new file called `/etc/bash_completion.d/htf`, eg.

```
echo "eval $(htf complete)" >> ~/.bashrc
. ~/.bashrc
```

or

```
echo "eval $(htf complete)" > /etc/bash_completion.d/htf
. /etc/bash_completion.d/htf
```

To check if bash-completion is enabled type

```
complete -p | grep htf
```

Open docs

To open the current docs in your webbrowser, type

```
htf docs
```

View licenses

To view the included and used open-source licenses, type

```
htf licenses
```

or

```
htf licenses --compact
```

to print the compact form.

To output the contents in Markdown format, type

```
htf licenses --markdown
```

or

```
htf licenses --markdown --compact
```

to print the compact form.

Show version

To show the `htf` version, type

```
htf version
```

htf.main() — Testscript utility

`htf.main` is used to start tests if not using `htf` (the command-line-utility) within a test-script.

```
htf.main(title: str = 'Testreport', tests: Any | List[Any] = '__main__', tags: str | None =
None, fixture_tags: str | None = None, html_report: str | List[str] | None = None,
junit_xml_report: str | List[str] | None = None, json_report: str | List[str] | None = None,
yaml_report: str | List[str] | None = None, report_server: str | List[str] | None = None,
extra_report: Any | List[Any] | None = None, create_drafts: bool = False, open_report: bool =
False, pattern: str = 'test*.py', metadata: Dict[str, Any] | None = None, parameters:
Dict[str, str] | None = None, exit: bool = True, fail_fast: bool = False,
minimize_report_size: bool = False, interactive: bool = False, interactive_address: str =
'0.0.0.0', interactive_port: int = 8080, open_browser: bool = True, shuffle: bool = False,
shuffle_seed: str | int | None = None) → int
```

Test runner.

Parameters:

- **title** – the test's title
- **tests=None** – a test-specifier or a list of test-specifiers (folder, file, test-case, test-case-method, module)
- **tags=None** – test filter expression
- **fixture_tags=None** – fixture filter expression
- **html_report=None** – HTML-report filename(s)
- **junit_xml_report=None** – JUnit-XML-report filename(s)
- **json_report=None** – JSON-report filename(s)
- **yaml_report=None** – YAML-report filename(s)
- **report_server=None** – Url or list of urls to report server
- **extra_report=None** – Extra reports that support instance.render(data). This is useful for `htf.DOORSTestReport` for example that needs additional parameters.
- **create_drafts=False** – if set to `True` draft reports are created after each test
- **open_report=False** – if set to `True` the HTML test report is opened after the test run
- **pattern="test*.py"** – the pattern to be used when searching for python files in folders
- **metadata=None** – a dictionary that contains metadata about the test run
- **parameters=None** – a dictionary that contains parameters for the test run
- **exit=True** – if set to `True` the exit() is called at the end of the tests
- **fail_fast=False** – if set to `True` the test run ends after the first failing test
- **minimize_report_size=False** – if set to `True` the report size is minimized.
- **interactive=False** – if set to `True` the tests are run interactively
- **interactive_address="0.0.0.0"** – the ip address the interaction server listens on.
- **interactive_port=8080** – the port the interaction server listens on.
- **open_browser=True** – if set to `False` browser is not opened automatically
- **shuffle=False** – set to `True` to shuffle tests
- **shuffle_seed=None** – the shuffle seed or 'last' to use the last shuffle seed

Returns:
`0` if the run was successful or the number of failed tests in case or errors and failures.
Return type:
int

Running tests

This section show some usage examples.

The first example covers the basic usage for a simple test-case running on Python 3.9+.

```
import htf


class ExampleTestCase(htf.TestCase):
    def test_example(self):
        self.assertTrue(True)
```

```
if __name__ == "__main__":
    htf.main()
```

This runs all tests within `ExampleTestCase` and generates an HTML-report names `testreport.html`.

Specifying tests

`htf.main` supports the `tests` parameter. It may be `None`, a test-specifier or a list of test-specifiers.

A test-specifier may be a python module, a feature file, a globbing-expression for python modules (e.g. `test_*.py`), a globbing expression for feature files (e.g. `features/*.feature`), a folder or an import-string (python style). In comparison to the `htf` command-line-utility it may also be a module-instance, a test-case or a test-case-method.

Test-specifiers can be mixed.

```
htf.main(tests=["<test-specifier1>", "<test-specifier2>", .. "<test-specifierN>"])
```

If only one test-specifier is supplied, it is not necessary to use a list

```
htf.main(tests="<test-specifier>")
```

If a test-specifier is a python module it is imported and all testable objects are collected into a test-suite that is used for the test run.

```
htf.main(tests="test_1.py")
```

If a test-specifier ist a globbing-expression the expression is evaluated and all matching files are imported.

```
htf.main(tests="test_*.py")
```

If a test-specifier is a folder this folder is searched recursively and all python modules found within (see File pattern, `pattern=`) are imported and all testable objects are collected into a test suite that is used for the test run.

Consider the folder `tests`, which contains python modules with unit tests. To execute all test cases in the folder `tests`, simply use

```
htf.main(tests="tests")
```

In case `tests` is an importable package or module, located in the python search path, the names might collide. In this case, simply use

```
htf.main(tests="tests/")
```

If a test-specifier is an import string, this may be a package, a module, a class (inheriting from `htf.TestCase`) or a method.

If a test-specifier is a package or a module, it is imported and containing tests are executed.

```
htf.main(tests=["tests", "tests.test_example"])
```

If a class is used it should be a testcase inheriting from `htf.TestCase`. This testcase is used in a test suite and all tests are run.

```
htf.main(tests=tests.test_example.ExampleTestCase)
```

If a method is used it should be a method within a test case.

```
htf.main(tests="tests.test_example.ExampleTestCase.test_one")
```

The examples above are analog to the command-line-utility `htf`.

In contrast to the command-line-utility htf, `htf.main` supports even more test-specifiers.

To test a module-instance use

```
import htf
import htf_examples.htf_main_1

if __name__ == "__main__":
    htf.main(tests=htf_examples.htf_main_1)
```

To test a test-case use

```
import htf


class ExampleTestCase(htf.TestCase):
    def test_example(self):
        self.assertTrue(True)
```

```
if __name__ == "__main__":
    htf.main(tests=ExampleTestCase)
```

And to test a test-case-method use

```
import htf


class ExampleTestCase(htf.TestCase):
    def test_example1(self):
        self.assertTrue(True)

    def test_example2(self):
        self.assertTrue(True)

if __name__ == "__main__":
    htf.main(tests=ExampleTestCase.test_example1)
```

Warning

When specifying methods as test specifiers in Python 3 the test class will be the class implementing the method and not necessarily the one specified. This means the methods called will also be the ones of the implementing class and may not be the ones you expect.

To clarify see the following example.

```
class Super:
    def whoami(self):
        print("Super")

    def test_whoami(self):
        self.whoami()


class Sub(Super):
    def whoami(self):
        print("Sub")
```

When running htf with `Sub.test_whoami` as a test specifier `Super.test_whoami` is executed instead. This is due to a limitation in Python 3 where the inheriting class can not be resolved from the function reference.

```
htf.main(tests=Sub.test_whoami)
```

To work around this issue you can either use the class itself as a specifier `Sub`, or pass an instance instead:

```
htf.main(tests=Sub().test_whoami)
```

Test-title

To specify the test-title use the `title=` argument.

```
htf.main(title="This is my first test")
```

HTML-report

By default `htf.main` generates an HTML-report called `testreport.html`. To specify an HTML-report use the `html_report=` argument.

```
htf.main(html_report="tests.html")
```

Multiple reports can be generated by supplying a list of filenames

```
htf.main(html_report=["tests1.html", "tests2.html"])
```

To open all HTML-reports use `open_report=True`.

```
htf.main(html_report="tests.html", open_report=True)
```

JUnit-XML-report

`htf.main` is also able to generate JUnit-XML-compatible XML-reports. The reports can be evaluated by different tools, eg. Jenkins.

To specify an XML-report use the `junit_xml_report=` argument.

```
htf.main(junit_xml_report="tests.xml")
```

or with multiple reports

```
htf.main(junit_xml_report=["tests1.xml", "tests2.xml"])
```

JSON-report

`htf.main` can also generate JSON-reports.

To specify a JSON-report use the `json_report=` argument.

```
htf.main(json_report="tests.json")
```

or with multiple reports

```
htf.main(json_report=["tests1.json", "tests2.json"])
```

YAML-report

`htf.main` can also generate YAML-reports.

To specify a YAML-report use the `yaml_report=` argument.

```
htf.main(yaml_report="tests.yaml")
```

or with multiple reports

```
htf.main(yaml_report=["tests1.yaml", "tests2.yaml"])
```

Other reports

There are other reports that need more parameters so that they need to be instantiated outside of `htf.main`.

For example `htf.DOORSTestReport` needs more information about report-module, requirements-modules and links-modules.

To use such a report use the `extra_report=` argument

```
import htf


class Tests(htf.TestCase):

    def test_assertionError(self):
        o = None
        print(o.no_there)

if __name__ == "__main__":
    doors_report = htf.DOORSTestReport(
                        filename="doors.dxl",
                        reportModule="/path/to/report/module",
                        requirementsModules={
                            "REQ_.*": "/path/to/requirements_module",
                            "TEST_.*": "/path/to/test_specification",
                        },
                        linksModules={
                            "REQ_.*": "/path/to/links/to/requirements_module",
                            "TEST_.*": "/path/to/links/to/test_specification",
                            }
                        )
    htf.main(extra_report=doors_report)
```

You can also supply a list of report instances

```
htf.main(extra_report=[report1, report2, ..., reportN])
```

Report Server

To send test results to a test report server (hre) use `report_server`.

```
htf.main(report_server="https://reports.host/project/token")
```

To disable SSL-verfification you can set the environment variable `HTF_INSECURE` to any value.

Draft Reports

To create draft reports after each test set `create_drafts=True`.

```
htf.main(create_drafts=True)
```

Minimized Report Size

To minimize the report size use `minimize=True`. When used for example captured stdout and stderr is only kept for failed tests. By default the report size is not minimized.

```
htf.main(minimize=True)
```

Shuffle Tests

To shuffle tests before running use `shuffle=True`.

```
htf.main(shuffle=True)
```

To set the shuffle seed use `shuffle_seed=<int>|'last'`. If `'last'` is used the seed is loaded from `'shuffle_seed.txt'`. If the file cannot be found one seed is created.

After a seed is created it is stored to be used next time. Using the seed the shuffle result can be reproduced.

```
htf.main(shuffle=True, shuffle_seed=12345)
htf.main(shuffle=True, shuffle_seed='last')
```

Filename-templates

Filenames for reports can include prepopulated templates.

Available templates:

- `{{title}}` - the title as a filename
- `{{date}}` - the current date `"%Y-%m-%d_%H-%M-%S"`
- `{{hostname}}`, `{{host}}` or `{{node}}` - the hostname
- `{{htf_version}}` or `{{version}}` - the htf version, eg. `htf_1.0.0`
- `{{python_version}}` - the python version, eg. `Python_3.6.1`

To generate an HTML-report containing the current node and the current date

```
htf.main(html_report="test_{{host}}_{{date}}.html")
```

Fail fast

If the `fail_fast=True` argument is used the test run ends after the first failing test.

File pattern

If a test-specifier is a folder the default pattern used for file discovery is `test*.py`. The pattern can be changed using the `pattern=` argument.

```
htf.main(tests="tests/", pattern="*.py")
```

Tagging

To select tests using a logical expression use the `tags` option set to an expression.

```
htf.main(tags="foo|bar")
```

To list all available tags use the `htf.get_tags` function.

```
htf.get_tags(tests="/path/to/test")
```

```
htf.get_tags(tests: Any | List[Any] = '__main__', pattern: str = 'test*.py') → List[str]
```

Get a list of tags found in the specified tests.

Parameters:

- **tests=None** – a test-specifier or a list of test-specifiers (folder, file, test-case, test-case-method, module)
- **pattern="test*.py"** – the pattern to be used when searching for python files in folders

Returns:
a list of tags found in the specified tests.
Return type:
list of str

To use tags for fixtures use the `fixture_tags` option set to an expression.

```
htf.main(fixture_tags="hardware")
```

Metadata

To add metadata about the test run use the `metadata` parameter set to a dictionary containing metadata.

This feature can be used to, for example, add information about a tested firmware version, etc.

```
metadata_dict = dict(firmware_version="1.2.3", hardware_version="4.5.6")
htf.main(metadata=metadata_dict)
```

The metadata will appear in the test reports.

Parameters

You can add parameters to test runs to control the behavior of tests and fixtures. Use the `parameters` parameter as a dict to supply parameters to test runs.

The parameters are available via the `htf.fixtures.Parameters` fixture.

```
htf.main(parameters=dict(name1="value1", name2="value2"))
```

The parameters will also appear in the test result metadata.

htf.run()

`htf.run` is an alias for `htf.main`.

```
htf.run(title: str = 'Testreport', tests: Any | List[Any] = '__main__', tags: str | None =
None, fixture_tags: str | None = None, html_report: str | List[str] | None = None,
junit_xml_report: str | List[str] | None = None, json_report: str | List[str] | None = None,
yaml_report: str | List[str] | None = None, report_server: str | List[str] | None = None,
extra_report: Any | List[Any] | None = None, create_drafts: bool = False, open_report: bool =
False, pattern: str = 'test*.py', metadata: Dict[str, Any] | None = None, parameters:
Dict[str, str] | None = None, exit: bool = True, fail_fast: bool = False,
minimize_report_size: bool = False, interactive: bool = False, interactive_address: str =
```

```
'0.0.0.0', interactive_port: int = 8080, open_browser: bool = True, shuffle: bool = False,
shuffle_seed: str | int | None = None) → int
```

Test runner.

Parameters:

- **title** – the test's title
- **tests=None** – a test-specifier or a list of test-specifiers (folder, file, test-case, test-case-method, module)
- **tags=None** – test filter expression
- **fixture_tags=None** – fixture filter expression
- **html_report=None** – HTML-report filename(s)
- **junit_xml_report=None** – JUnit-XML-report filename(s)
- **json_report=None** – JSON-report filename(s)
- **yaml_report=None** – YAML-report filename(s)
- **report_server=None** – Url or list of urls to report server
- **extra_report=None** – Extra reports that support instance.render(data). This is useful for `htf.DOORSTestReport` for example that needs additional parameters.
- **create_drafts=False** – if set to `True` draft reports are created after each test
- **open_report=False** – if set to `True` the HTML test report is opened after the test run
- **pattern="test*.py"** – the pattern to be used when searching for python files in folders
- **metadata=None** – a dictionary that contains metadata about the test run
- **parameters=None** – a dictionary that contains parameters for the test run
- **exit=True** – if set to `True` the exit() is called at the end of the tests
- **fail_fast=False** – if set to `True` the test run ends after the first failing test
- **minimize_report_size=False** – if set to `True` the report size is minimized.
- **interactive=False** – if set to `True` the tests are run interactively
- **interactive_address="0.0.0.0"** – the ip address the interaction server listens on.
- **interactive_port=8080** – the port the interaction server listens on.
- **open_browser=True** – if set to `False` browser is not opened automatically
- **shuffle=False** – set to `True` to shuffle tests
- **shuffle_seed=None** – the shuffle seed or 'last' to use the last shuffle seed

Returns:
`0` if the run was successful or the number of failed tests in case or errors and failures.
Return type:
int

htf.verify()

`htf.verify` verifies the system environment for validated installations.

```
htf.verify() → bool
```

Verify system environment. Needs `htf-validated` installed and the license feature `verification`.

Returns:
`True` in case the system environment was successfully verified else `False`
Return type:
bool

Docker Integration

Introduction

Since `htf-1.3` HILSTER supplies multi platform Docker containers for every version that can be used like the `htf` command line utility. The great advantage is that the user does not need to install and update HILSTER Testing Framework manually because Docker does this job automatically.

For example to download the latest version into Docker simply run

```
docker pull hilster/htf
```

Operation Systems

HILSTER offers Docker containers for Linux and Windows (both amd64) for the latest supported Python version and every patch release.

This documentation assumes that Windows hosts run Windows Docker containers. It is also possible to run Linux Docker containers from a Windows host. In this case the examples will not work without modification.

Tags

HILSTER supports tags for every patch version (e.g. `hilster/htf:1.3.1`, for every newest minor version (e.g. `hilster/htf:1.3`) and for every major version (e.g. `hilster/htf:1`).

The `hilster/htf:latest` or `hilster/htf` tags always point to the latest release.

We suggest to use a minor version and not the patch version to be able to automatically update to the newest patch version.

Volumes

The Docker containers have one volume to store licences. On Linux the volume is `/licenses` and on Windows it is `C:/licenses`. The volumes can also be bind mounted to the host filesystem to persist licenses.

Bind Mounts

The following bind mounts `/code` (Linux) or `C:/code` (Windows) and `/tests` (Linux) or `C:/tests` (Windows) can be used for testing.

`code` is used to supply user editable code for a project.

`tests` is used to supply user editable tests. `tests` is the working directory of the Docker containers.

PYTHONPATH

The environment variable `PYTHONPATH` includes the `code` and `tests` volumes so Python will automatically search for modules there.

Activation

To activate use the Docker only `--activate <activationkey>` option, e.g.:

```
docker run hilster/htf --activate AAAA-BBBB-CCCC-DDDD
```

The `--activate <activationkey>` option can also be added to all other options to `htf` and will be filtered out by a wrapper script that handles the activation. This way it is possible to automatically activate a node for every run for example in a continuous integration system.

Renewal

To renew licenses use the Docker only `--renew` option, e.g.:

```
docker run hilster/htf --renew
```

Interactive Mode

To run `htf` in Docker interactively enter

```
docker run hilster/htf --shell
```

SSL Verification

In some circumstances it might be necessary to disable SSL verification. To do this simply set the environment variable `HLM_INSECURE` to any value and activate or renew.

```
docker run -e HLM_INSECURE=1 hilster/htf --activate AAAA-BBBB-CCCC-DDDD
```

Command Line Utility

`hilster/htf` is used like a command line utility and uses the `htf` executable as the entry point. So the usage is exactly the same like for the `htf` command line utility.

Simply replace `htf <options>` with `docker run -it [...] hilster/htf <options>`.

You might also like to create an alias for `docker run -it [...] hilster/htf` named `htf` to be able to run `htf` in Docker like they way it would be installed locally.

Community Edition

Docker is also available for the community edition. Simply replace the tag `htf` with `htf-community`.

```
docker run hilster/htf-community
```

Examples

Running Tests

On Linux run

```
docker run -it -v $(pwd)/project:/code -v $(pwd)/tests:/tests hilster/htf
```

and on Windows run

```
docker run -it -v %cd%/project:C:/code -v %cd%/tests:C:/tests hilster/htf
```

Running Tests with Activation and persistently stored license

Lets assume you have activation key `AAAA-BBBB-CCCC-DDDD` and the node shall renew its licenses or activate itself.

Linux

Activation:

```
docker run -it \
    -v $(pwd)/project:/code \
    -v $(pwd)/tests:/tests \
    -v $(pwd)/licenses:/licenses \
    hilster/htf --activate AAAA-BBBB-CCCC-DDDD
```

Run tests with persistently stored license without activation:

```
docker run -it \
    -v $(pwd)/project:/code \
    -v $(pwd)/tests:/tests \
    -v $(pwd)/licenses:/licenses \
    hilster/htf
```

Windows

Activation:

```
docker run -it ^
    -v %cd%/project:C:/code ^
    -v %cd%/tests:C:/tests ^
    -v %cd%/licenses:C:/licenses ^
    hilster/htf --activate AAAA-BBBB-CCCC-DDDD
```

Run tests with persistently stored license without activation:

```
docker run -it ^
    -v %cd%/project:C:/code ^
    -v %cd%/tests:C:/tests ^
    -v %cd%/licenses:C:/licenses ^
    hilster/htf
```

The `--activate <activation key>` option is not an `htf` option but is handled by a wrapper script.

Running Specific Tests

Lets assume you have `tests.initial_tests` as an importable module bind mounted to `tests` and you only want to run these tests.

On Linux run

```
docker run -it -v $(pwd)/project:/code -v $(pwd)/tests:/tests hilster/htf initial_tests
```

and on Windows run

```
docker run -it -v %cd%/project:C:/code -v %cd%/tests:C:/tests hilster/htf initial_tests
```

Using Docker for a Project with Dependencies

Often you need to add some dependencies for your project stored in `requieements.txt`. If you need requirements and want to use Docker for testing you can create your own Docker container based on `hilster/htf`.

To do this create a file `requirements.txt` with your requirements for example

```
numpy
matplotlib
scipy
```

Create a `Dockerfile` with contents like

```
FROM hilster/htf
MAINTAINER HILSTER <mail@hilster.io>

ADD requirements.txt requirements.txt
RUN pip install -r requirements.txt
```

This will add `requirements.txt` to your container and will install the dependencies while being built.

To build and tag your Docker container simply run

```
docker build -t <project_name> .
```

To run your container simply run

```
docker run -it -v $(pwd)/project:/code -v $(pwd)/tests:/tests <project_name>
```

on Linux and

```
docker run -it -v %cd%/project:C:/code -v %cd%/tests:C:/tests <project_name>
```

on Windows.

Mapping Comports

To map a comport on a Linux host use the `--device` option.

The following example shows how to map `/dev/ttyUSB0` to the Docker container.

```
docker run -it --device=/dev/ttyUSB0 hilster/htf
```

On Windows devices have a different description in form of `<IdType>/><Id>`.

The following example maps a comport to the Docker container.

```
docker run -it --device="class/86E0D1E0-8089-11D0-9CE4-08003E301337" hilster/htf
```

Mapping USB Dongles

On Linux map `/sys/bus/usb/devices` into the Docker container and use the `--privileged` option to be able to use USB dongles from within the container.

```
docker run -it --privileged -v /sys/bus/usb/devices:/sys/bus/usb/devices hilster/htf
```

On Windows you have to add the corresponding USB device to be handled by the Docker container.

Dry Run

The dry run feature is supported by the HILSTER Testing Framework since version `htf-1.2`.

The dry run feature works like a normal test run but marks all tests to be skipped and is fast as a result.

This is useful feature if you want check your test selection with or without tagging. It can also be used to easily create a test specification if your test specifications are included in the docstring of your tests.

The test specification can be read within an HTML-report or you can use a JSON-report to create another document using the contained data.

htf.dryrun()

`htf.dryrun` can be used to do a dry run of a test suite from a Python script. It accepts the same parameters as `htf.main` and `htf.run` do and so it can easily be substituted.

```
htf.dryrun(title: str = 'Testreport', tests: Any | List[Any] = '__main__', tags: str | None =
None, fixture_tags: str | None = None, html_report: str | List[str] | None = None,
junit_xml_report: str | List[str] | None = None, json_report: str | List[str] | None = None,
yaml_report: str | List[str] | None = None, report_server: str | List[str] | None = None,
extra_report: Any | List[Any] | None = None, create_drafts: bool = False, open_report: bool =
False, pattern: str = 'test*.py', metadata: Dict[str, Any] | None = None, parameters:
Dict[str, str] | None = None, exit: bool = True, fail_fast: bool = False,
minimize_report_size: bool = False, interactive: bool = False, interactive_address: str =
'0.0.0.0', interactive_port: int = 8080, open_browser: bool = True, shuffle: bool = False,
shuffle_seed: str | int | None = None) → int
```

Test runner for a dry run to watch the test suite.

Parameters:

- **title** – the test's title
- **tests=None** – a test-specifier or a list of test-specifiers (folder, file, test-case, test-case-method, module)

- **ags=None** – test filter expression
- **fixture_tags=None** – fixture filter expression
- **html_report=None** – HTML-report filename(s)
- **junit_xml_report=None** – JUnit-XML-report filename(s)
- **json_report=None** – JSON-report filename(s)
- **yaml_report=None** – YAML-report filename(s)
- **report_server=None** – Url or list of urls to report server
- **extra_report=None** – Extra reports that support instance.render(data). This is useful for `htf.DOORSTestReport` for example that needs additional parameters.
- **create_drafts=False** – if set to `True` draft reports are created after each test
- **open_report=False** – if set to `True` the HTML test report is opened after the test run
- **pattern="test*.py"** – the pattern to be used when searching for python files in folders
- **metadata=None** – a dictionary that contains metadata about the test run
- **parameters=None** – a dictionary that contains parameters for the test run
- **exit=True** – if set to `True` the exit() is called at the end of the tests
- **fail_fast=False** – if set to `True` the test run ends after the first failing test
- **minimize_report_size=False** – if set to `True` the report size is minimized.
- **interactive=False** – if set to `True` the tests are run interactively
- **interactive_address="0.0.0.0"** – the ip address the interaction server listens on.
- **interactive_port=8080** – the port the interaction server listens on.
- **open_browser=True** – if set to `False` browser is not opened automatically
- **shuffle=False** – set to `True` to shuffle tests
- **shuffle_seed=None** – the shuffle seed or 'last' to use the last shuffle seed

Returns:
`0` if the run was successful or the number of failed tests in case or errors and failures.
Return type:
int

htf dryrun

To dry run from command line use `htf dryrun ...`. It accepts the same parameters as `htf run` does and so it can easily be substituted.

Tagging

Tagging allows you to attach arbitrary labels to testcases and tests which can then be selected using a logical expression.

Tagging is a very useful feature to group tests or to partition and paralell long running test suites easily without any additional effort.

Adding tags to testcases and tests

Tagging is done using the `htf.tags` decorator which can be used for testcases and tests. Tests inherit the tags from the testcase they are located in. Inherited tags are merged with local tags.

```
import htf

@htf.tags("foo")
class TestFoo(htf.TestCase):

    @htf.tags("one")
    def test_one(self):
        pass

    @htf.tags("two")
    def test_two(self):
        pass

    @htf.tags("three", "io")
    def test_three(self):
        # use io
        pass

@htf.tags("bar")
class TestBar(htf.TestCase):

    @htf.tags("one")
    def test_one(self):
        pass

    @htf.tags("two")
    def test_two(self):
        pass

    @htf.tags("three", "io")
    def test_three(self):
        # use io
        pass
```

All tests in `TestFoo` are tagged with `"foo"` and all tests in `TestBar` are tagged with `"bar"`.

Logical expression to select tags

Tagged tests can be selected using a logical expression that is evaluated to find matching tests.

A logical expression is a string consisting of tag names, operators and pairs of brackets.

The parser supports the following tokens:

Token	Description
tag	A string matching the regular expression `r'[a-zA-Z0-9_\-\.\=]+'`. So it consists of `a..z`, `A-Z` `0..9`, `_`, `-`, `=` and `.`.
&	Logical and
|	Logical or
^	Logical xor
!	Logical not
(	Left bracket
)	Right bracket

Groups are built using the brackets. Groups are evaluated first.

Whitespaces are ignored.

For example to select all tests that have the `"foo"` tag the logical expression would be `"foo"`.

To select all tests that do not have the `"foo"` tag the logical expression would be `"!foo"`.

To select all tests that uses `"io"` and `"bar"` the expression is `"io&bar"` (or `"io & bar"` for better readability).

To select all tests that are tagged with `"bar"` or `"io"` and `"three"` the expression is `"bar | (io & three)"`.

Selecting tests using tags

To select tests using a logical expression and tags is easy.

`htf.main`, `htf.run` and `htf.dryrun` have the `tags` option set to a logical expression.

For example:

```
htf.main(tags="bar | (io & three)")
```

For the command line utilities `htf run` and `htf dryrun` use the `-t` or `--tags` option followed by a logical expression.

For example:

```
htf run --tags='bar | (io & three)'
```

Showing available tags

To get a list of available tags you can use `$ htf tags` from the command line or `htf.get_tags` from within a Python script.

For the initial example this would lead to a sorted list of tags consisting of `bar`, `foo`, `io`, `one`, `three` and `two`.

Data Driven Testing

Data Driven Testing allows dynamic generation of tests using a data source. The data source can be Python objects, JSON, CSV or YAML files.

To use Data Driven Testing simply decorate tests using the data or combinatoric decorators as mentioned below.

Code-based Examples

Full Example

In the following example the test function is called three times with a value from a list of scalars. To use data driven testing in `htf` the test methods are decorated with the decorator supplying the data values.

```
import htf

@htf.data([1.0, 2.0, 3.0])
def test_scalar(value):
    print("value:", value)

if __name__ == "__main__":
    htf.main()
```

List of Scalars

The following examples demonstrate the usage of Python objects as data input. The decorator to be used when the data is supplied as a Python object is `@htf.data(<object>)`.

The following example uses a list of integers as data.

```
@htf.data([23, 24, 25])
def test_scalar(value):
    print("value:", value)

# generates calls
# test_scalar(value=23)
# test_scalar(value=24)
# ...
```

Iterator

An iterator can also be supplied.

```
@htf.data(iter([23, 24, 25]))
def test_iterator(value):
    print("value:", value)
```

Generator

It is also possible to supply a generator.

```
def generator() -> Generator[int, None, None]:
    yield 10
    yield 20
    yield 30
```

```
@htf.data(generator())
def test_generator_instance(value):
    print("value:", value)

# generates calls
# test_generator_instance(value=10)
# test_generator_instance(value=20)
# ...

@htf.data(generator)
def test_generator(value):
    print("value:", value)

# generates calls
# test_generator(value=10)
# test_generator(value=20)
# ...
```

Async generators are supported, too.

```
async def generator() -> AsyncGenerator[int, None]:
    yield 10
    yield 20
    yield 30

@htf.data(generator())
async def test_generator_instance(value):
    print("value:", value)
```

List of Dicts

The following example supplies a list of dictionaries. The number of parameters must match the number of dictionary entries.

```
@htf.data([dict(a=23, b=1), dict(a=24, b=2), dict(a=25, b=3)])
def test_dict(a, b):
   print("a:", a, "b:", b)

# generates calls
# test_dict(a=23, b=1)
# test_dict(a=24, b=2)
# ...
```

List of Lists

Lastly it is also possible to supply lists of lists. When called, `value` contains a list.

```
@htf.data([[1, 2, 3], [4, 5, 6]])
def test_list_of_int(value):
    print("value:", value)
```

```
# generates calls
# test_list_of_int(value=[1, 2, 3])
# ...

@htf.data([[dict(value=23)], [dict(value=24)], [dict(value=25)]])
def test_list_of_lists_of_dict(values):
   print("value:", values)

# generates calls
# test_list_of_lists_of_dict(value=23)
# test_list_of_lists_of_dict(value=24)
# ...
```

Unpack the content of `value` by setting `unpack=True` to distribute the list entries to three parameters.

```
@htf.data([[1, 2, 3], [4, 5, 6]], unpack=True)
def test_list_of_int_unpack(a, b, c):
    print("a:", a, "b:", b, "c:", c)

# generates
# test_list_of_int_unpack(a=1, b=2, c=3)
# ...
```

File-based Examples

The following examples demonstrate the usage of various file formats as data input. The supported file formats and their corresponding decorators are:

File type	Decorator
JSON-file	`@htf.json_data("filename.json")`
YAML-file	`@htf.yaml_data("filename.yml")`
CSV-file	`@htf.csv_data("filename.csv")`

In each case there is also the option of first creating an iterator for the file content which is handed to the `@htf.data()` decorator. For this purpose the following syntax is used:

File type	Decorator and Iterator
JSON-file	`@htf.data(htf.JSONFileIterator("filename.json"))`
YAML-file	`@htf.data(htf.YAMLFileIterator("filename.yml"))`
CSV-file	`@htf.data(htf.CSVFileIterator("filename.csv"))`

Furthermore, decorators for JSON and YAML input have the option `unpack` set to `False` by default. If changed to `True` lists will be unpacked as demonstrated in the below examples. There is no such option for the CSV decorator.

JSON Test Data

To supply simple scalar values as data from a JSON file, use the following code:

```
@htf.json_data("ddt_scalar.json")
def test_json_scalar(value):
    print("value: ", value)

# generates calls
# test_json(value=1337)
# test_json(value=1338)
# ...

@htf.data(htf.JSONFileIterator("ddt_scalar.json"))
def test_json_iterator(value):
    print("value:", value)

# generates calls
# test_json_iterator(value=1337)
# test_json_iterator(value=1338)
# ...
```

Contents of `ddt_scalar.json`:

```
[1337, 1338, 1339]
```

If the supplied data comes in form of nested lists, `unpack` can be set to `True` to assign each inner list element to a variable.

```
@htf.json_data("ddt_list.json", unpack=True)
def test_json_list_unpack(a, b, c):
    print("value: ", a, b, c)

# generates calls
# test_json_list(a=111, b=222, c=333)
# test_json_list(a=11, b=22, c=33)
# ...

@htf.json_data("ddt_list.json", unpack=False)
def test_json_list(value):
    print("value: ", value)

# generates calls
# test_json_list(value=[111, 222, 333])
# test_json_list(value=[11, 22, 33])
# ...
```

Contents of `ddt_list.json`:

```
[[1,2,3], [11,22,33], [111,222,333]]
```

YAML Test Data

Supplying scalar values as data from a YAML file works analogously to JSON files.

```
@htf.yaml_data("ddt_scalar.yml")
def test_yaml_scalar(value):
    print("value:", value)

# generates calls
# test_yaml(value=1337)
# test_yaml(value=1338)
# ...

@htf.data(htf.YAMLFileIterator("ddt_scalar.yml"))
def test_yaml_iterator(value):
    print("value:", value)

# generates calls
# test_yaml_iterator(value=1337)
# test_yaml_iterator(value=1338)
# ...
```

Contents of `ddt_scalar.yml`:

```
[1337, 1338, 1339]
```

The following code showcases the use of the `unpack` option in the case of a YAML file.

```
@htf.yaml_data("ddt_list.yml", unpack=True)
def test_yaml_list_unpack(a, b, c):
    print("value: ", a, b, c)

# generates calls
# test_yaml_list(a=111, b=222, c=333)
# test_yaml_list(a=11, b=22, c=33)
# ...

@htf.yaml_data("ddt_list.yml", unpack=False)
def test_yaml_list(value):
    print("value: ", value)

# generates
# test_yaml_list(value=[111, 222, 333])
# test_yaml_list(value=[11, 22, 33])
# ...
```

Contents of `ddt_list.yml`:

```
[[1,2,3], [11,22,33], [111,222,333]]
```

CSV Test Data

Tests can also be generated using CSV data sources. The first line of the CSV data is a header. The parameter names of the decorated method must match the entries of the header, i.e. the fist line of each column must contain the name of the parameter to which the data is to be assigned. Each line is supplied as a `dict`. Note that the entries of the `dict` are strings by default.

```
@htf.csv_data("ddt.csv")
def test_csv(a, b):
    print("a: ", a, "b: ", b)

# generates calls
# test_csv(a=1337, b=2222)
# a: 1337 b: 2222
# test_csv(a=1338, b=3333)
# a: 1338 b: 3333
# ...

@htf.data(htf.CSVFileIterator("ddt.csv"))
def test_csv_iterator(a, b):
    print("a: ", a, "b: ", b)

# generates calls
# test_csv(a=1337, b=2222)
# a: 1337 b: 2222
# test_csv(a=1337, b=3333)
# a: 1338 b: 3333
# ...
```

Contents of `ddt.csv`:

```
a,b
1337,2222
1338,3333
1339,4444
```

CSV data source values are not converted automatically. The desired conversion can be specified with the optional `convert_to` parameter. `convert_to` can be a callable or list thereof. If conversion fails with a ValueError the value is passed as a string.

```
@htf.csv_data("ddt.csv", convert_to=[int, float])
def test_csv(a, b):
    print("a: ", a, "b: ", b)

# generates calls
# test_csv(a=1337, b=2222.0)
# a: 1337 b: 2222.0
# test_csv(a=1338, b=3333.0)
# a: 1338 b: 3333.0
# ...
```

Combinatoric Data Generation

Combinatoric decorators can be used to manipulate the supplied data.

Cartesian Product

The `product` function produces every possible combination of items, taking one item of each given iterable.

```
@htf.product([1, 2, 3], repeat=2)
def test_product(a, b):
    print("a:", a, "b:", b)

# generates calls
# test_product(a=1, b=1)
# test_product(a=1, b=2)
# ...

@htf.product([1, 2], [3, 4], repeat=1)
def test_products(a, b):
    print("a:", a, "b:", b)

# generates calls
# test_products(a=1, b=3)
# test_products(a=2, b=3)
# ...
```

In combination with the aforementioned file iterator functions, this can be used to produce combinations of elements of given data files.

```
@htf.product(htf.JSONFileIterator("ddt_scalar.json"), repeat=2)
def test_json_products(a, b):
    print("a:", a, "b:", b)

# generates calls
# test_products(a=1337, b=1337)
# test_products(a=1337, b=1338)
# ...

@htf.product(list(range(10)), repeat=3)
def test_long_products(*args):
    print("args:", args)

# generates calls
# test_long_products(*(0, 0, 1))
# test_long_products(*(0, 0, 2))
# ...
```

Permutations

The `permutations` function creates all permutations of length `r` of a given iterable.

```
@htf.permutations([1, 2, 3], r=2)
def test_permutations(a, b):
    print("a:", a, "b:", b)

# generates calls
# test_permutations(a=1, b=2)
# test_permutations(a=1, b=3)
# ...
```

Combinations

The `combinations` function produces all possible combinations of elements of an iterable without changing their order. Elements are not combined with themselves.

```
@htf.combinations([1, 2, 3], r=2)
def test_combinations(a, b):
    print("a:", a, "b:", b)

# generates calls
# test_combinations(a=1, b=2)
# test_combinations(a=1, b=3)
# ...
```

The `combinations_with_replacement` function does the same as the combinations function, except elements are also combined with themselves.

```
@htf.combinations_with_replacement([1, 2, 3], r=2)
def test_combinations_with_replacement(a, b):
    print("a:", a, "b:", b)

# generates calls
# test_combinations_with_replacement(a=1, b=1)
# test_combinations_with_replacement(a=1, b=2)
# ...
```

Data Decorators

```
htf.data(iterable: Iterable, unpack: bool = False) → Callable[[Callable[[...], Any]], Any]
```

Create a data driven test using data.

Parameters:

- **iterable** – an iterable used as parameter data source. Can also be a callable.
- **unpack=False** – if set to `True` iterables are unpacked.

```
htf.csv_data(filename: str, convert_to: Iterable[Any] | Generator[Any, None, None] |
List[Any] | Any | None = None) → Callable[[Callable[[...], Any]], Any]
```

Create a data driven test using data from a CSV file. The first line of the CSV data is a header. Each column from the first is used as the parameter name of the decorated method. Each line is supplied as a `dict`.

Parameters:

- **filename** – the filename to be read.
- **convert_to** – data types to convert read values to (e.g. float, int)

```
htf.yaml_data(filename: str, unpack: bool = False) → Callable[[Callable[[...], Any]], Any]
```

Create a data driven test using data from a YAML file.

Parameters:

- **filename** – the filename to be read.
- **unpack=False** – if set to `True` iterables are unpacked.

```
htf.json_data(filename: str, unpack: bool = False) → Callable[[Callable[[...], Any]], Any]
```

Create a data driven test using data from a json file.

Parameters:

- **filename** – the filename to be read.
- **unpack=False** – if set to `True` iterables are unpacked.

Combinatoric Decorators

```
htf.product(*iterables: Iterable, **repeat: int) → Callable[[Callable[[...], Any]], Any]
```

Create a data driven test using `itertools.product` with a cartesian product of input `*iterables`.

Parameters:

- ***iterables** – iterables to be passed to `itertools.product`
- **repeat=1** – number of repetitions.

```
htf.permutations(iterable: Iterable, r: int | None = None) → Callable[[Callable[[...],
Any]], Any]
```

Create a data driven test using `itertools.permutations` with successive `r` length permutations of elements in the `iterable`.

Parameters:

- **iterable** – iterable to be passed to `itertools.permutations`
- **r=None** – length of permutations

```
htf.combinations(iterable: Iterable, r: int) → Callable[[Callable[[...], Any]], Any]
```

Create a data driven test using `itertools.combinations` with successive `r` length subsequences of elements from `iterable`.

Parameters:

- **iterable** – iterable to be passed to `itertools.combinations`
- **r** – length of combinations

```
htf.combinations_with_replacement(iterable: Iterable, r: int) → Callable[[Callable[[...],
Any]], Any]
```

Create a data driven test using `itertools.combinations` with successive `r` length subsequences of elements from `iterable` allowing individual elements to be repeated more than once.

Parameters:

- **iterable** – iterable to be passed to `itertools.combinations_with_replacement`
- **r** – length of combinations

File Iterators

```
class htf.CSVFileIterator(filename: str, convert_to: Iterable[Any] | Generator[Any, None,
None] | List[Any] | Any | None = None)
```

CSV file iterator.

Parameters:
filename – the filename to be read.
Returns:
iterator for the file contents as `dict`. convert_to: data types to convert read values to (e.g. float, int)
Return type:
iterator

```
class htf.YAMLFileIterator(filename: str)
```

YAML file iterator.

Parameters:
filename – the filename to be read.
Returns:
iterator for the file contents.

Return type:
iterator

```
class htf.JSONFileIterator(filename: str)
```

JSON file iterator.

Parameters:
filename – the filename to be read.
Returns:
iterator for the file contents.
Return type:
iterator

Interactive Testing

In some circumstances not all tests can be completely automated.

htf offers interactive testing which allows you to interact with your tests via your web browser. A web server is started on port `8080`.

A QR code is printed on the terminal so that you can interact using your mobile phone or a tablet easily.

This feature can be used to script and revision manual tests for example.

Interaction is done via the `interaction` fixture.

htf needs to run in interactive mode. When running from command line add the `-i` argument or set `interactive=True` in `htf.main`.

By default a browser is opened to be used for interaction. If you do not wish to open the browser automatically use `--no-browser` or pass `open_browser=False` to `htf.main`.

To set the listening address use `--interactive-address` or `interactive_address` in `htf.main` To set the port to listen on use `--interactive-port` or `interactive_port` in `htf.main`.

All interaction dialogs will be put into the HTML test report with the appropriate answers.

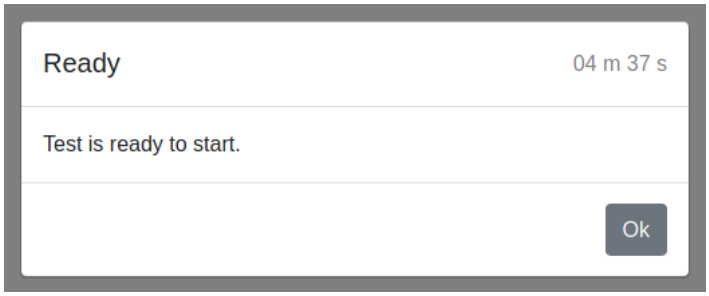
Interaction Dialogs

Message Dialog

To inform the user you can create a `message_dialog`. The message dialog has an ok button that will hide it immediately.

```
def test_message_dialog(interaction):
    """
    This test shows how to use a message dialog.
```

```
    Args:
        interaction (htf.fixtures.interaction): the interaction fixture
    """
    interaction.message_dialog("Ready", "Test is ready to start.")
```

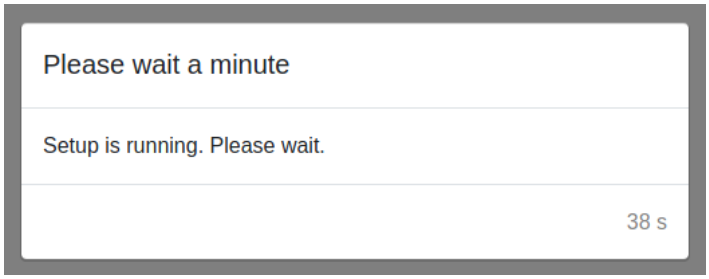


Waiting Dialog

To let the user wait for a given time you can use `waiting_dialog`. The timeout must not be `None`.

```
def test_waiting_dialog(interaction):
    """
    This test shows how to use a waiting dialog.

    Args:
        interaction (htf.fixtures.interaction): the interaction fixture
    """
    interaction.waiting_dialog("Please wait a minute", "Setup is running. Please wait.", timeout=60)
```

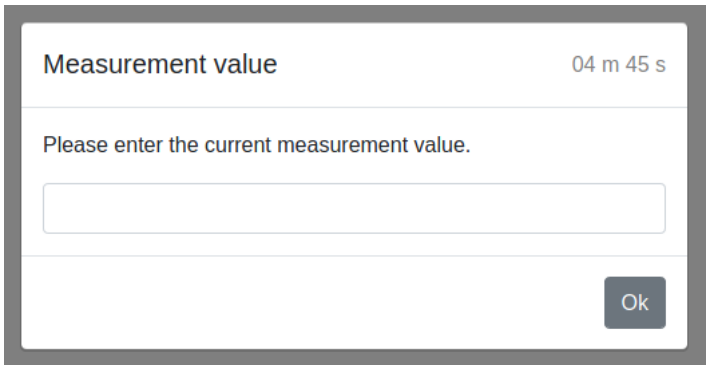


Input Dialog

If your test needs user input you can use `input_dialog`.

```
def test_input_dialog(interaction):
    """
    This test shows how to use an input dialog.

    Args:
        interaction (htf.fixtures.interaction): the interaction fixture
    """
    value = interaction.input_dialog("Measurement value", "Please enter the current measurement
value.")
    # value is a string
```



You can optionally validate the user's inputs on server side that only valid inputs are accepted. The validator can also act as a type cast.

```
def test_input_dialog(interaction):
    """
    This test shows how to use an input dialog with a float validator.

    Args:
        interaction (htf.fixtures.interaction): the interaction fixture
    """
    value = interaction.input_dialog("Measurement value", "Please enter the current measurement
value.",
                           validator=htf.float_validator)
    # value is a float
```

There are some built-in validators.

```
htf.int_validator(value: Any) → int
```

An int validator for interactive tests.

```
htf.float_validator(value: Any) → float
```

A float validator for interactive tests.

```
htf.regex_validator(pattern: str, exception_message: str | None = None) → Callable[[str],
str]
```

A regex validator factory.

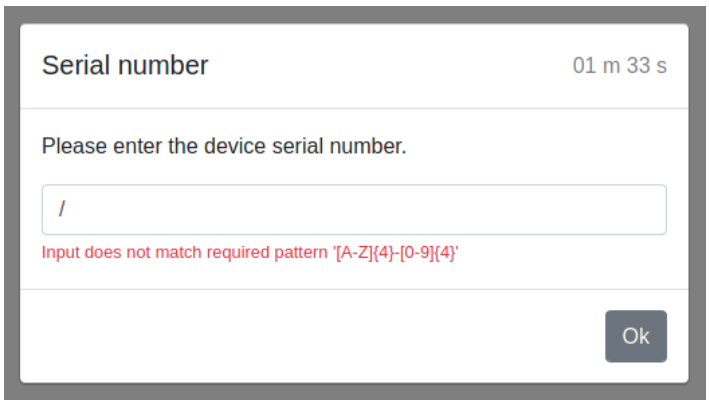
Parameters:

- **pattern** – the regex pattern to be checked
- **exception_message=None** – the optional exception message supplied to the user in its browser

Returns:
an interaction validator for pattern

```
def test_input_dialog(interaction):
    """
    This test shows how to use an input dialog with a regex validator.

    Args:
        interaction (htf.fixtures.interaction): the interaction fixture
    """
    value = interaction.input_dialog("Serial number", "Please enter the device serial number.",
                         validator=htf.regex_validator(pattern=r"[A-Z]{4}-[0-9]{4}"))
    # value is a valid serial number
```

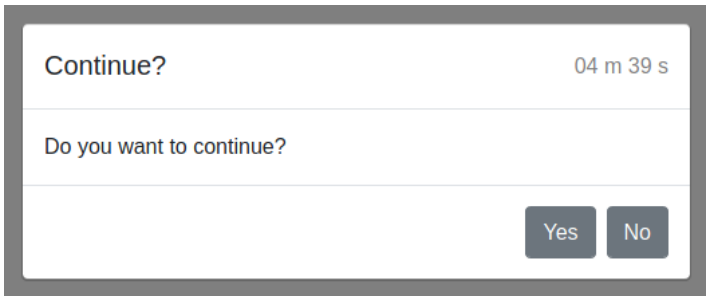


Yes-No Dialog

To ask the user a yes-no question you can use `yes_no_dialog`.

```
def test_yes_no_dialog(interaction):
    """
    This test shows how to use a yes-no dialog.

    Args:
        interaction (htf.fixtures.interaction): the interaction fixture
    """
    value = interaction.yes_no_dialog("Continue?", "Do you want to continue?")
    # value is 'yes' or 'no' or None in case of a timeout
```

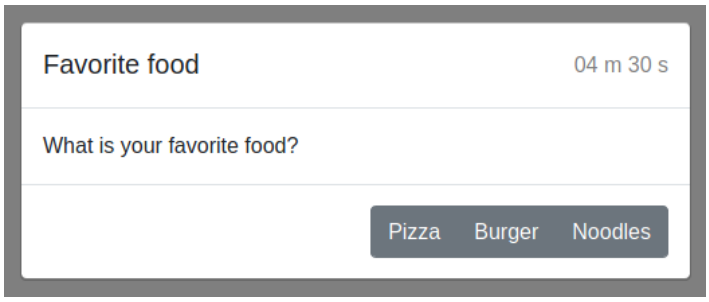


Button Dialog

To ask the user a question with one answer out of multiple predefined choices you can use the `button_dialog`.

```
def test_button_dialog(interaction):
    """
    This test shows how to use a button dialog.

    Args:
        interaction (htf.fixtures.interaction): the interaction fixture
    """
    value = interaction.button_dialog("Favorite food", "What is your favorite food?",
                                      choices=[
                                          ("Pizza", "pizza"),
                                          ("Burger", "burger"),
                                          ("Noodles", "noodles"),
                                      ])
    # value is 'pizza', 'burger', 'noodles' or None in case of a timeout
```

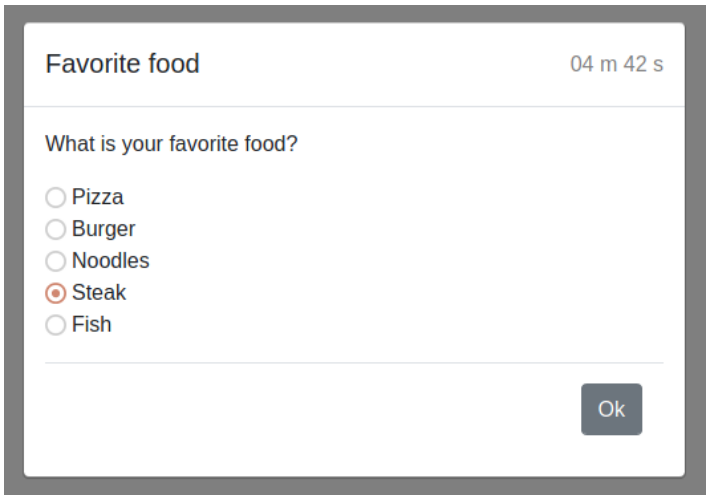


Radio Button Dialog

To ask the user a question with one answer out of multiple predefined choices you can use the `radio_button_dialog`.

```
def test_radio_button_dialog(interaction):
    """
    This test shows how to use a radio button dialog.

    Args:
        interaction (htf.fixtures.interaction): the interaction fixture
    """
    value = interaction.radio_button_dialog("Favorite food", "What is your favorite food?",
                choices=[
                    "Pizza",  # label and answer are the same
                    ["Burger", "burger"],  # label is left and answer is right
                    ("Noodles", "noodles"),  # supplied as a tuple
                    ["Steak", True],  # label and answer are the same and the checkbox is checked
                    ["Fish", "fish", False]
#  # label is left and answer is right and the checkbox is not checked
                ])
    # value is 'Pizza', 'burger', 'noodles', 'Steak', 'fish' or None in case of a timeout
```

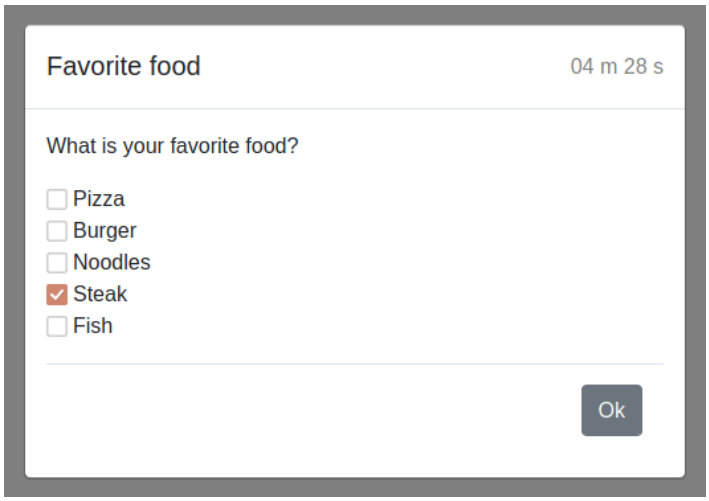


Checkbox Dialog

To ask the user a question with multiple predefined answers you can use the `checkbox_dialog`.

```
def test_checkbox_dialog(interaction):
    """
    This test shows how to use a checkbox dialog.

    Args:
        interaction (htf.fixtures.interaction): the interaction fixture
    """
    value = interaction.checkbox_dialog("Favorite food", "What is your favorite food?",
                choices=[
                    "Pizza",  # label and answer are the same
                    ["Burger", "burger"],  # label is left and answer is right
                    ("Noodles", "noodles"),  # supplied as a tuple
                    ["Steak", True],  # label and answer are the same and the checkbox is checked
                    ["Fish", "fish", False]
#  # label is left and answer is right and the checkbox is not checked
                ])
    # value is a list containing no elements or multiple choices from
    # 'Pizza', 'burger', 'noodles', 'Steak', 'fish' or None in case of a timeout
```

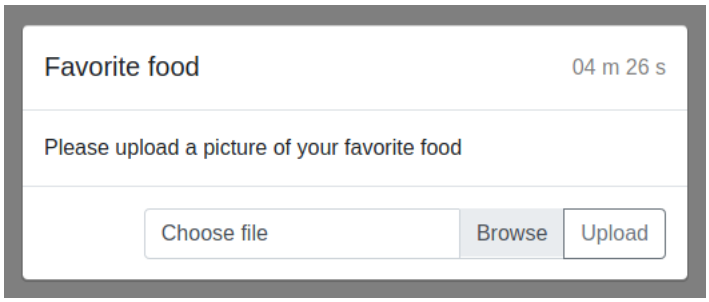


File Upload Dialog

To allow the user to upload a file you can use the `file_upload_dialog`.

```
def test_file_upload_dialog(interaction):
    """
    This test shows how to use a file_upload dialog.

    Args:
        interaction (htf.fixtures.interaction): the interaction fixture
    """
    value = interaction.file_upload_dialog("Favorite food", "Please upload a picture of your
favorite food", ".png, image/*")
    # value is a tuple( file name, file type, uploaded file in binary form) or None in case of a
timeout.
```



Captcha Functionality

Using the `captcha` library allows you to simulate Captcha functionality in dialogs.

```
def test_solve_captchas(interaction, assertions):
    """
    This test shows how to interactively solve captchas.

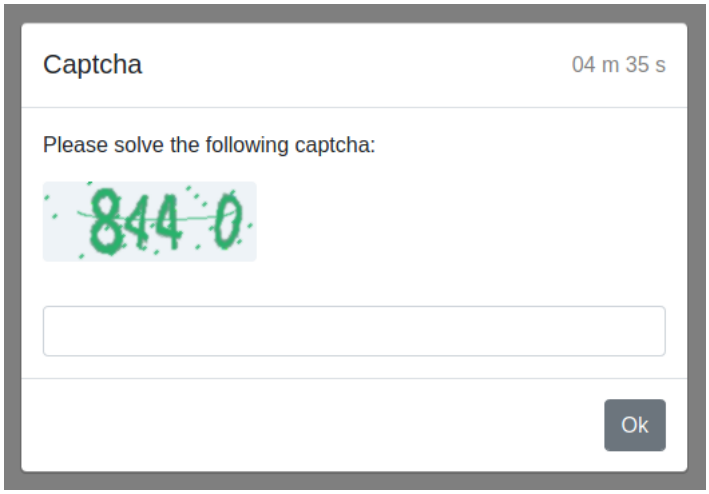
    Args:
         interaction (htf.fixtures.interaction): the interaction fixture
         assertions (htf.fixtures.assertions): the assertions fixture
    """
    try:
        for number_solved in range(1, 6):  # max 5 captchas can be solved
            image = ImageCaptcha()
            captcha_value = str(randint(1000, 10000))
            image.write(captcha_value, 'captcha.png')

            captcha = """

            .. |captcha| image:: captcha.png
            """

            answer = interaction.input_dialog(title='Captcha',
                                              text="Please solve the following captcha:\n\n|
captcha|" + captcha)
            assertions.assert_equal(answer, captcha_value)
            if number_solved < 5:
                again = interaction.yes_no_dialog(title='Again',
```

```
                                                text="Would you like to solve another one?")
                if again == 'no':  # stop if the user has had enough
                    break
    except AssertionError:
        number_solved -= 1
        raise
    finally:
        interaction.message_dialog(title='Congratulations', text="You solved {}
captchas!".format(number_solved))
```



Embed Images in Dialogs

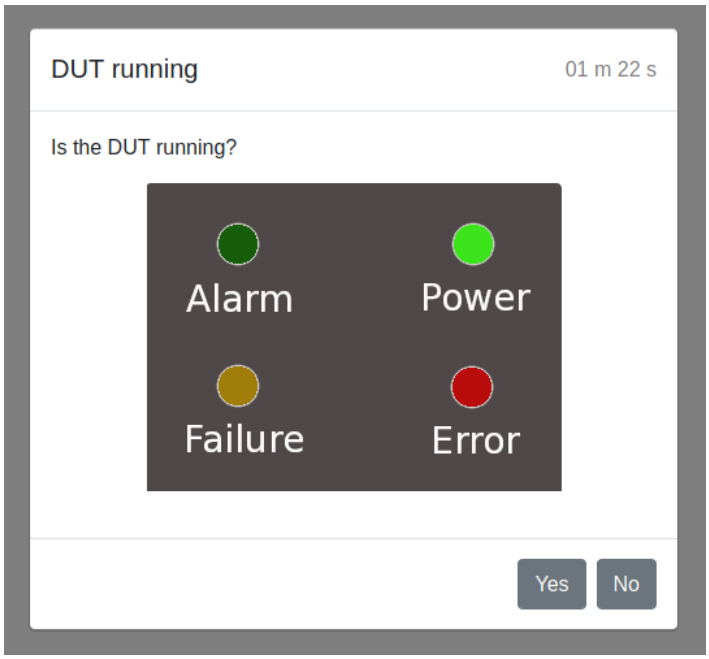
The `text` parameter of all dialogs can be reStructuredText. It is possible to embed images that appear in the dialog and in the HTML test report.

Images can be embedded as a paragraph or inline.

Images as Paragraphs

To add an image as a paragraph use

```
.. image:: path/to/image.png
```

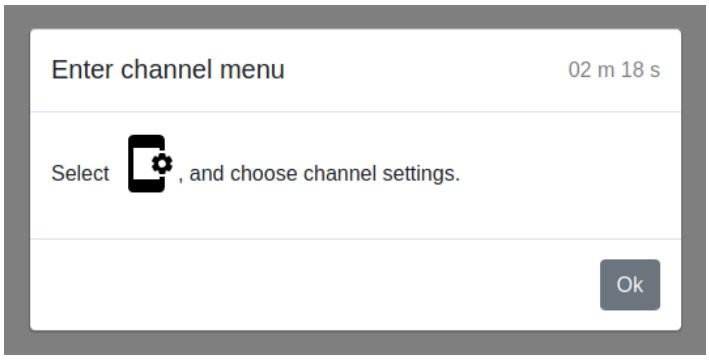


Inline Images

To add an image inline use substitution definitions.

```
This |inlineimage| is located inline.

.. |inlineimage| image:: path/to/image.png
```



Repeat or Cancel Tests

On the top right corner in the browser a test control panel can be found. Users can repeat or cancel a test by clicking the corresponding button and entering a message.

> **Note**
>
> Cancelling or repeating a test is performed in the same thread in which the test is executed. Thus, a test can block the reaction (e.g. if it runs `time.sleep` or any other blocking call). The task of waiting for a longer time should therefore be split into several shorter calls to `time.sleep`, etc.

Reference

```
class htf.fixtures.interaction(interaction_server: BrowserInteractionServer, result:
TestResult)
```

```
button_dialog(title: str, text: str, choices: List[str] | Tuple[str], timeout: float =
300) → Any
```

Create a dialog with buttons and definable answers.

Parameters:

- **title** – the title of the button dialog
- **text** – the content of the button dialog
- **choices** – each choice can be a string containing the label and the answer a tuple or list of two strings where the first entry is the label and the second label is the answer.
- **timeout=300** – the timeout until the button dialog disappears in seconds (`None` means no timeout)

Returns:
the answer for the clicked button or `None` in case of a timeout
Return type:
object or None

```
checkbox_dialog(title: str, text: str, choices: List[str] | Tuple[str], timeout: float =
300) → Any
```

Create a dialog with definable checkboxes for multiple choices.

Parameters:

- **title** – the title of the checkbox dialog
- **text** – the content of the checkbox dialog
- **choices** – each choice can be a string containing the label and the answer a tuple or list of two strings where the first entry is the label and the second label is the answer. You can also supply a list or tuple with two entries where the first entry is the label and the second entry is a boolean that states if the radio button is checked. You can also supply a list or tuple with three entries where the first entry is the label, the second entry is the answer and the third entry is a boolean that states if the radio button is checked.
- **timeout=300** – the timeout until the checkbox dialog disappears in seconds (`None` means no timeout)

Returns:
a list containing the selected answers or `None` in case of a timeout
Return type:
list of str or None

```
file_upload_dialog(title: str, text: str, file_types: str | None = None, timeout: float
= 300) → File | None
```

Create a dialog which allows the user to upload a file.

Parameters:

- **title** – the title of the file upload dialog
- **text** – the content of the file upload dialog
- **file_types** – desired file type of files to be uploaded. This should be respected by the file chooser in all modern browsers. File types can be specified as file extensions or MIME types, with every entry separated by a comma.
- **timeout=300** – the timeout until the file upload dialog disappears in seconds (`None` means no timeout)

Returns:
tuple of str or tuple or None. `( file name, file type, uploaded file in binary form)`.

```
input_dialog(title: str, text: str, timeout: float = 300, validator: Callable[[...], Any]
| None = None) → Any
```

Create an input dialog to let the user interact with some textual input.

Parameters:

- **title** – the title of the input dialog
- **text** – the content of the input dialog
- **timeout=300** – the timeout until the input dialog disappears in seconds (`None` means no timeout)
- **validator=None** – the validator to validate the inputs on server side

Returns:
the string the user supplied, the return value of the validator or `None`
in case of a timeout

Return type:
object or None

```
message_dialog(title: str, text: str, timeout: float = 300) → Any
```

Create an interactive message dialog to display a message.

Parameters:

- **title** – the title of the message dialog
- **text** – the content of the message dialog:
- **timeout=300** – the timeout until the message dialog disappears in seconds (`None` means no timeout)

Returns:
`"ok"` in case of a click on the ok button or `None` in case of a timeout.

```
radio_button_dialog(title: str, text: str, choices: List[str] | Tuple[str], timeout:
float = 300) → Any
```

Create a dialog with definable radio buttons to select one answer out of many choices.

Parameters:

- **title** – the title of the radio button dialog
- **text** – the content of the radio button dialog
- **choices** – each choice can be a string containing the label and the answer a tuple or list of two strings where the first entry is the label and the second label is the answer. You can also supply a list or tuple with two entries where the first entry is the label and the second entry is a boolean that states if the radio button is checked. You can also supply a list or tuple with three entries where the first entry is the label, the second entry is the answer and the third entry is a boolean that states if the radio button is checked.
- **timeout=300** – the timeout until the radio button dialog disappears in seconds (`None` means no timeout)

Returns:
the answer for the selected radio button or `None` in case of a timeout
Return type:
str or None

```
waiting_dialog(title: str, text: str, timeout: float) → Any
```

Create a waiting dialog.

Parameters:

- **title** – the title of the waiting dialog
- **text** – the content of the waiting dialog
- **timeout** – the timeout until the waiting dialog disappears in seconds

Returns:
`'Ok'` after the timeout is over.

```
yes_no_dialog(title: str, text: str, timeout: float = 300) → Any
```

Create a yes-no dialog to ask the user a question.

Parameters:

- **title** – the title of the yes-no dialog
- **text** – the content of the yes-no dialog
- **timeout=300** – the timeout until the yes_no_dialog disappears in seconds (`None` means no timeout)

Returns:
`'yes'` or `'no'` depending on the user input or `None` in case of a timeout
Return type:
str or None

Settings

`htf.Settings` is used to store and modify local settings that can easily be used within your test cases or components. The local settings are stored in JSON or YAML format. JSON is the default format. YAML is assumed if the settings file extension is `.yml` or `.yaml`.

If no `filename` parameter is given and no `htf.Settings` instance exists the settings file is searched for using the following algorithm.

First the environment variable `$HTF_SETTINGS` is checked. If it does not exist or settings cannot be loaded, settings are searched for in `$HOME` and the current working directory. The expected file name is `settings` followed by the extensions `.txt`, `.json`, `.yml` or `.yaml`.

The first file that exists is loaded. If it contains errors no settings are loaded and an exception is raised.

`htf.Settings` behaves like a dictionary. So you can read and write values by keys, update it and iterate items. It is also possible to access keys without indexing like these were normal attributes

```
assert settings["key"] == settings.key  # True
```

It can be pretty-printed, too to inspect its contents.

`htf.Settings` is a singleton. This means that the settings are loaded once and every other created instance is always the same.

If you want distinct instances use `htf.SettingsNoSingleton` which offer the same behaviour without being a singleton.

JSON Example

A JSON settings file named `settings.json` and the contents

```
{
    "comport": "/dev/ttyUSB0",
    "debug_flags": {
        "filename": "debug.txt",
        "level": 1
    },
    "last_updated": 1552207883,
    "report_server": "reports.hilster.io"
}
```

can easily be loaded with

```
s = htf.Settings(filename="settings.json")
print(s)
# <htf.Settings filename='settings.yml' data={
# "comport": "/dev/ttyUSB0",
# "debug_flags": {
#     "filename": "debug.txt",
#     "level": 1
# },
# "last_updated": 1552207883,
```

```
# "report_server": "reports.hilster.io"
# }>
```

To access members use

```
s.comport  # or
s["comport"]
```

To set a single member use

```
s.another_comport = "/dev/ttyUSB1"  # or
s["another_comport"] = "/dev/ttyUSB1"
```

To update the settings with another dictionary or other settings use

```
s.update(dict(a=1, b=2))
print(s)
# <htf.Settings filename='settings.json' data={
# "a": 1,
# "b": 2,
# "comport": "/dev/ttyUSB0",
# "debug_flags": {
#     "filename": "debug.txt",
#     "level": 1
# },
# "last_updated": 1552207883,
# "report_server": "reports.hilster.io"
# }>
```

To save settings use

```
s.save()
```

YAML Example

A YAML settings file named `settings.yml` and the contents

```
# This is a comment within a settings file that JSON does not support
comport: "/dev/ttyUSB0"
debug_flags:
    filename: "debug.txt"
    level: 1
last_updated: 1552207883
report_server: "reports.hilster.io"
```

can easily be loaded with

```
s = htf.Settings(filename="settings.yml")
print(s)
# <htf.Settings filename='settings.yml' data={
# "comport": "/dev/ttyUSB0",
# "debug_flags": {
#     "filename": "debug.txt",
#     "level": 1
# },
# "last_updated": 1552207883,
# "report_server": "reports.hilster.io"
# }>
```

Attention: when saving YAML all comments are lost!

Create a Settings File

To create new settings use

```
s = htf.Settings(
        filename="new_settings.yml",
        create="True",
        defaults={
            "foo": "bar",
        })
# save settings
s.save()
# cat new_settings.yml
# foo: "bar"
```

htf.Settings

```
class htf.Settings(*args: Any, **kwargs: Any)
```

`Settings` can be used to access local settings easily. It is a singleton.

The local settings file is automatically searched looking in `$HTF_SETTINGS`, `$HOME/settings.{txt,json,yml,yaml}` or `./settings.{txt,json,yml,yaml}` in this order. If no settings file was found an exception is raised except `create` is `True`.

Settings are stored in JSON or YAML format. If the file extension is `.yml` or `.yaml` YAML data is assumed else JSON.

Instantiate a Settings instance.

Parameters:

- **filename=None** – the filename to load explicitly

- **create=False** – if set to `True` no exception is raised if no settings are found to create new settings
- **defaults=None** – if set to a dict these values are used as default values before loading the settings file

```
get(key: str, default: Any | None = None) → Any
```

Get an item and return `default` if the item was not found.

Parameters:

- **key** – key for item
- **default=None** – is returned if `key` was not found.

Returns:
self[key]
Return type:
object

```
get_filename() → str | None
```

Return the currently used settings filename.

```
get_settings_filenames() → List[str]
```

Return a list of possible settings filenames that are tried in order. The first available file is used. You can overwrite this method in your specialized `Settings`.

Returns:
a list of filenames that are tried to be opened from left to right.
The first file that is opened will be used to read settings from.

Return type:
list

```
items() → Iterator[Any]
```

Iterate over items like in a dict.

Returns:
Iterator over the items (key, value)

```
save() → None
```

Save current settings to current filename (useful after modification).

Attention: YAML comments are lost when overwriting an existing YAML file!

```
update(data: Dict[str, Any]) → None
```

Update settings with `data`.

Parameters:
data – data to use for update

htf.SettingsNoSingleton

```
class htf.SettingsNoSingleton(*args: Any, **kwargs: Any)
```

Settings without being a singleton.

Instantiate a Settings instance.

Parameters:

- **filename=None** – the filename to load explicitly
- **create=False** – if set to `True` no exception is raised if no settings are found to create new settings
- **defaults=None** – if set to a dict these values are used as default values before loading the settings file

```
get(key: str, default: Any | None = None) → Any
```

Get an item and return `default` if the item was not found.

Parameters:

- **key** – key for item
- **default=None** – is returned if `key` was not found.

Returns:
self[key]
Return type:
object

```
get_filename() → str | None
```

Return the currently used settings filename.

```
get_settings_filenames() → List[str]
```

Return a list of possible settings filenames that are tried in order. The first available file is used. You can overwrite this method in your specialized `Settings`.

Returns:
a list of filenames that are tried to be opened from left to right.
The first file that is opened will be used to read settings from.

Return type:
list

```
items() → Iterator[Any]
```

Iterate over items like in a dict.

Returns:
Iterator over the items (key, value)

```
save() → None
```

Save current settings to current filename (useful after modification).

Attention: YAML comments are lost when overwriting an existing YAML file!

```
update(data: Dict[str, Any]) → None
```

Update settings with `data`.

Parameters:
data – data to use for update

Decorators

Decorators can be used to change the behaviour of a method by *decorating* it.

Decorators that are included in htf are:

htf.test

Decorate function or method to be interpreted as a test.

```
@htf.test
def function():
    pass
```

```
htf.test(func: Callable[[...], Any]) → Callable[[...], Any]
```

Mark a function or method as a test.

htf.after

Run additional commands after the test function.

```
def run_after(*args, **kwargs):
    print("run_after", args, kwargs)

@after(run_after, 1, 2, 3, a="b")
def function():
    print("function")
function()
# prints:
# function
# run_after (1, 2, 3) {'a': 'b'}
```

```
htf.after(after_function: Callable[[...], Any], *after_args: Any, **after_kwargs: Any) →
Callable[[Callable[[...], Any]], Any]
```

The *@after* decorator decorates a method so that *after_function* is called after the decorated function is called itself.

Parameters:

- **after_function** – the function to be run after the decorated function.
- ***after_args** – positional arguments to be passed to `afterFunction`.
- ****after_kwargs** – keyword-arguments to be passed to `afterFunction`.

htf.before

Run additional commands before the test function.

```
def run_before(*args, **kwargs):
    print("runBefore", args, kwargs)

@before(run_before, 1, 2, 3, a="b")
def function():
    print("function")
function()
# prints:
# run_before (1, 2, 3) {'a': 'b'}
# function
```

```
htf.before(before_function: Callable[[...], Any], *before_args: Any, **before_kwargs: Any) →
Callable[[Callable[[...], Any]], Any]
```

The *@before* decorator decorates a method so that *before_function* is called before the decorated function is called itself.

Parameters:

- **before_function** – the function to be run before the decorated function.
- ***before_args** – positional arguments to be passed to `beforeFunction`.
- ****before_kwargs** – keyword-arguments to be passed to `beforeFunction`.

htf.enqueue_exception

Decorate a method to fetch its exception that is enqueued into an exception queue.

```
@enqueue_exception(queue)
def decorated_method():
    raise Exception("This is a test")
```

```
htf.enqueue_exception(queue: Queue) → Callable[[Callable[[...], Any]], Any]
```

Decorate a method to fetch its exception that is enqueued into an exception queue. The enqueued exception can be handled elsewhere. The caught exception can be raised after enqueueing it.

Just decorate you method with the queue to be used and every exception that is raised within the decorated method will be put into the queue before the exception is raised again.

Parameters:
queue – the queue to enqueue the exception.

htf.metadata

Provides a way to set metadata in the test report for the decorated function.

```
@htf.metadata(name="Bob", age="20")
def test_with_metadata():
    pass
```

```
htf.metadata(**metadata: Any) → Callable[[...], Any]
```

Provides a way to set metadata in the test report for the decorated function.

Parameters:
metadata – the dictionary of metadata

htf.requirements

Used to show that a test checks a certain requirement.

```
@requirements("REQ_1", "REQ_2")
def test_with_requirements():
    pass
```

```
htf.requirements(*requirements: str) → Callable[[Callable[[...], Any]], Any]
```

The `@requirements` decorator reports which requirements are checked by a test in the test report shown in the `HTMLTestReport`.

Parameters:
***requirements** – a tuple of strings that name the checked requirements, eg. “REQ_1”, “REQ_2”, etc.

htf.implements

Used to annotate a test with implemented tests. These links are used to generate links in test reports and to send test results back to ALM/QMS systems.

```
@implements("TEST-1", "TEST-2")
def test_implementing_test_specifications():
    pass
```

```
htf.implements(*tests: str) → Callable[[Callable[[...], Any]], Any]
```

The `@implements` decorator reports which tests are implemented.

Parameters:
***tests** – A tuple of strings that name the implemented tests, eg. “TC-1”, “TC-2”, etc. You can also pass a url, eg. “http://server/path/to/testcase” where “testcase” is the id

htf.meets_doors_requirements

To use in conjunction with the `doors_testreport` to show that a test checks a certain requirement.

```
@meets_doors_requirements("doors://host:port/path-without-id-",
                  "REQ_1", "REQ_2")
def test_with_a_useful_name():
    pass
```

```
htf.meets_doors_requirements(base_url: str, *requirements: str) → Callable[[Callable[[...],
Any]], Any]
```

The `@meets_doors_requirements` decorator reports which requirements are checked by a test in the test report shown in the `doors_testreport` and the `HTMLTestReport`.

Parameters:

- **base_url** – the DOORS base url the links are created with.
- ***requirements** – a tuple of strings that name the checked requirements, eg. “REQ_1”, “REQ_2”, etc.

htf.periodic

Run a decorated function periodically.

```
@periodic(period=1.0)
def periodically_called_every_second():
    print("call")
```

`@periodic` can also be used within classes

```
class ClassWithAPeriodicMethod():

    @periodic(period=1.0)
    def periodically_called_every_second(self):
        print("call")
```

```
htf.periodic(period: float, maximum_period: float | None = None, run_condition:
Callable[[...], bool] | None = None, raise_exception: bool = True) →
Callable[[Callable[[...], Any]], Any]
```

Decorate a method to be run periodically.

Parameters:

- **period** – the period in seconds.
- **maximum_period=None** – if set to a float >= *period* the periodic method may take up to *maximum_period* time without raising an exception. A warning is printed on stdout instead.
- **run_condition=None** – a callable (method or lambda expression) that has to return true while the method is run periodically.
- **raise_exception=True** – if set to `True` an exception is raised if the called method takes longer than the period (or `maximum_period` if set).

Warning

Non-realtime operating systems will not ensure timing accuracy.

htf.raises

The decorated function will catch an exception

```
@raises(AssertionError)
def test_failure(assertions):
```

```
    assertions.assert_true(False)

@raises(TimeoutException, UnknownException)
def test_exceptions():
    raise TimeoutException("timeout")
    raise UnknownException("unknown exception")
```

```
htf.raises(*exceptions: BaseException) → Callable[[Callable[[...], Any]], Any]
```

Decorate a method to catch different exceptions.

Parameters:
***exceptions** – a tuple of exceptions that are fetched.

htf.skip

Decorate to skip the test method or function.

```
@skip("This test is skipped in view of the occasion")
def test_always_skipped():
    pass
```

```
htf.skip(reason: str) → Callable[[Callable[[...], Any]], Any]
```

Decorate a method to be skipped in view of the occasion.

Parameters:
reason – the reason to be put into the test report.
Raises:
SkipTest – to skip the test.

htf.skip_if

Decorate to skip test if condition evaluates to `True` at runtime. Condition is called with the same parameters like the decorated test including all fixtures, etc. and if it failes it is called without parameters.

```
def skip_tests():
    return True

@htf.skip_if(skip_tests, "Skipped if skip_tests returns True")
def test_skipped_if():
    pass

@htf.skip_if(True, "Always skipped")
def test_always_skipped():
    pass
```

```
@htf.skip_if(False, "Never skipped")
def test_never_skipped():
    pass

@htf.skip_if(lambda: True, "Always skipped")
def test_always_skipped_lambda():
    pass

def condition():
    return True

@htf.skip_if(condition, "skipped")
def test_always_skipped_with_bound_method(self):
    pass

class SkipTests(htf.TestCase):
    def condition(self, *args, **kwargs):
        # can only be evaluated at runtime
        return True

@htf.skip_if(condition, "classed skipped")
def test_skipped_by_instance_condition(self, assertions):
    pass

def ddt_condition(a):
    return a < 100

@htf.data(range(3))
@htf.skip_if(ddt_condition, "skipped ddt")
def test_ddt_skipped(a):
    pass

def fixture_condition(assertions):
    return True

@htf.skip_if(fixture_condition, "skipped with fixtures")
def test_skipped_with_fixtures(assertions):
    pass

def catch_all_condition(*args, **kwargs):
    return True

@htf.skip_if(catch_all_condition, "skipped with args and kwargs")
def test_args_kwargs(assertions, delay, step):
    pass
```

```
htf.skip_if(condition: bool | Callable[[...], bool], reason: str) →
Callable[[Callable[[...], Any]], Any]
```

Decorate a method to be skipped. The `condition` is evaluated at runtime. Note that `condition` is evaluated twice by the test runner and the test itself. Thus, it should not introduce any side effects.

Parameters:

- **condition** – the condition may be `True`, `False` or a callable object that must return `True` or `False`. The decorated test is skipped if `condition` is `True` or returns `True`.
- **reason** – the reason to be put into the test report.

Raises:
SkipTest – to skip the test if `condition` is or returns `True`.

htf.stacktrace

Every time the decorated function is called, the call stack will be printed.

```
@stacktrace
def function():
    pass

function()  # will print(the call stack to stdout)
```

```
htf.stacktrace(func: Callable[[...], Any]) → Callable[[...], Any]
```

The `@stacktrace` decorator decorates a method so that its call stack is printed to stdout every time it is called. This is useful for debugging.

htf.tags

Attach a tag to tests, fixtures or classes.

```
htf.tags(*tags: str) → Callable[[Callable[[...], Any]], Any]
```

Tag a test or class.

Parameters:
***tags** – the tags to be used

htf.timed

Ensure ending of a method or function within a given time limit.

```
@timed(limit=0.1)
def decorated_method_that_fails():
    time.sleep(0.2)

@timed(limit=1.0)
def decorated_method_that_does_not_fail():
    time.sleep(0.2)
```

```
htf.timed(limit: float) → Callable[[Callable[[...], Any]], Any]
```

Decorate a method to ensure that it ends within a given time limit.

Parameters:
limit – the time limit in seconds.

User-Defined Decorators

Users can add their own decorators for their own code.

Since htf supports coroutines, functions and methods as tests, user-defined decorators need to support those as well.

Note that some decorators deliver a coroutine regardless of the decorated object.

Therefore, the decorator needs to check whether the decorated object is a coroutine or not and call it accordingly.

```
import asyncio
from functools import wraps

def user_decorator() -> Callable[[Callable[..., Any]], Any]:
    def wrapper(func: Callable[..., Any]) -> Callable[..., Any]:
        @wraps(func)
        async def _user_decorator_wrapper(*args: Any, **kwargs: Any) -> Any:
            # user code
            if asyncio.iscoroutinefunction(func):
                await func(*args, **kwargs)  # call a coroutine
            else:
                func(*args, **kwargs)  # call everything that is not a coroutine
            # user code
        return _user_decorator_wrapper
    return wrapper

@user_decorator()
@htf.requirements("REQ-1")
def test_example():
    pass

@user_decorator()
@htf.requirements("REQ-1")
async def test_example_async():
    pass
```

Test Reports

HILSTER Testing Framework includes test reports for HTML, XML, YAML and JSON.

HTML Test Report

The HTML test report is a monolithic Javascript application that does not need to load any external data.

Historical HTML test-report walkthrough

Download recording · Download transcript

```
Historical HTML test-report walkthrough
=======================================

Visual walkthrough of the silent original screen recording, based on reviewed
overview/detail/filter/search/attachment/print frames. This records the historical
interface, not a newly executed test run or current product validation.

Original recording: showcase_testreport.mp4

1. Open the report overview. A doughnut chart and result bar summarize six tests:
   three successful tests, one failure, one error, and one skipped test. Hovering
   the chart highlights the successful half and displays its count/percentage.

2. Open an error result. The expanded red result card displays an exception,
   traceback, timestamps/duration, documentation, metadata and captured stdout.

3. Browse the All and Success result lists from the sidebar. The All list contains
   docstring, error, failure, plot/attachment, skip and CAPTCHA examples; Success
   narrows the list to the three successful tests.

4. Use the search box to filter for attach. Matching text is highlighted in the
   plot/attachments result. Its expanded green card shows the documentation, tags,
   successful Create plot/Attach plot steps, metadata, stdout and a Lorenz Attractor
   image attachment. Clear the search to return to the result list.

5. Open the printable report and scroll through it. The print view contains the
   HILSTER title page, overview and individual test details, including the docstring
   example with table/text content and the error example with its traceback.
```

When loading the test report, a progress bar appears, and the percentage of loaded tests is shown. The HTML test report usually loads very fast.

When the test report has finished loading, a responsive interface is shown, allowing you to interact with the test report.

Navigation

The navigation is located in the sidebar to the left. The first two entries, `Overview` and `Metadata`, are helpful for getting an initial impression of the test run. When the test report is loaded, `Overview` is shown by default, visualizing and summarizing the test results in an easy to understand way. `Metadata` shows test run specific information.

The remaining entries are used for navigating through the test results, and filtering which test results to display. Clicking `All` shows all test results. When selecting result types (`Success`, `Error`, `Failure` and `Skipped`), the test results get filtered accordingly. It is possible to select multiple result types at once. The number of tests within the respective result types are shown next to the filters.

Note that clicking `All`/pressing `esc` resets any applied filters.

Information about the host, the platform the tests were run on and licensing information are shown at the bottom of the sidebar on the left.

The sidebar can be hidden by clicking on the hamburger icon on the top, next to the logo.

Test title and test date are shown in the top right corner.

Overview

The visualisation is hierarchical: The inner ring represents the the four results types `success`, `error`, `failure` and `skipped`, as shown in the sidebar. The next ring shows more detailed information, such as exception types for failures and errors, and skip reasons for skipped tests. These information get even more detailed in the outer ring, making it possible to easily identify concrete exception strings within respective exception types.

Hovering over the diagram reveals additional information, such as hierarchy paths and statistics.

Clicking a section of the diagram searches the test report for the respective result type/exception type/exception string.

Test results

The main content is located to the right of the sidebar, containing all test results. Clicking on a result expands the respective test and shows further information. By default all test results are collapsed.

Every test result contains different test properties, such as `result`, `traceback`, `metadata`, `documentation`, `tags`, `dates`, `attachments`, captured `stdout` and captured `stderr`. To improve performance, long output (`stdout` and `stderr`) is not loaded by default. Clicking on `show more` loads the respective output.

File attachments

The HTML test report supports file attachments that you can attach to a test using `htf.TestCase.attach_file` (or `htf.TestCase.attachFile`).

File attachments are not supported by all browsers. For more information see Browser Limitations.

URL attachments

You can also attach URLs by using `htf.TestCase.attach_url` (or `htf.TestCase.attachUrl`) in your test.

Searching

Clicking on the magnifier icon on the top opens a search interface. The HTML test report offers three different search methods: Default search, global search and special pattern search.

Note that only selected categories get searched. In order to search all testresults, you would need to select `All` in the sidebar.

The search interface can be closed by clicking on the small `x` to the right of the search bar.

Default search

By default, only test titles get searched. Typing in the search bar instantly triggers the search.

Toggling the regex slider, located to the right side of the search bar, allows the user to perform a search using regular expressions.

The test report also offers two additional advanced searching methods, global search and special pattern search.

Global search

Global search requires the search term to be prefixed with `global:`, and allows the user to search not only titles, but also in any test properties within the test results, such as `Dates`, `Traceback` or `Tags`.

For example to search for tests that include `foo`, the user would use `global:foo`.

To search for tests that include `foo` or `bar`, you would use `global:foo|bar`, combining global search with regular expressions.

Special pattern search

Special pattern search allows to search in specific test properties, such as `Tags`, `Dates` or `Traceback`, by using the name of the respective property as prefix in lowercase. For example, to search for tests tagged with `tag_1`, one would search for `tags:tag_1`. Accordingly, searching for tests which were started on `2020-01-09`, one would search for `dates:2020-01-09`.

This can also be used to filter for all tests containing a specific property. For example, to only show tests containing `Tags`, one would search for `tags:`.

Goto

When pressing `g`, an input field appears next to the search bar, allowing the user to jump to a specific test. Entering the test run number (which can be shown by pressing `n`) of the respective test and pressing `Enter` expands and focuses the test.

Bookmarking

Right-clicking a test bookmarks the test for easier access. A bookmark-section is shown in the sidebar when there are active bookmarks.

Draft

Even when not all tests are run yet, the test report can be interacted with. This is indicated by a red `draft` sign in the sidebar.

Printing

While in test run overview, a print icon is shown in the upper right corner. Clicking it will render a print optimized version of the test report, including an overview diagram, the test run metadata and a signature page.

In print view, every test will get expanded and printed on a dedicated page for easier readability.

Dark Mode

For easier readability, the testreport color scheme can be changed to a darker variant by pressing `D`. Pressing the hotkey again switches back to the light theme.

Hotkeys

The HTML test report supports the following hotkeys:

Key	Description
`h`	Show help (any click or key but `h` hides it again)
`esc`	Reset view: hide help, hide search box, show all results
`a`	Collapse all expanded test results
`o`	Show test run overview diagram
`m`	Show test run metadata
`a`	Show all results
`s`	Toggle successes
`f`	Toggle failures
`e`	Toggle errors
`S`	Toggle skipped
`F`	Show search interface
`R`	Show search interface and toggle regex parameter
`g`	Show goto interface to jump to a specific test
`n`	Toggle test run numbers
`b`	Toggle sidebar
`l`	Show open source licenses
`p`	Show printable version of test report
`D`	Activate Dark Mode

The only hotkey you have to remember is `h` though.

Browser Limitations

Most modern browsers, such as Firefox, Chrome and Microsoft Edge (Version 80+), support the whole functionality of the test report.

File attachments are realized using data-urls.

Some Microsoft-browsers do not support them at all. Chrome and Chromium have security features in place, but support an acceptable workaround.

The following browser limitations are known:

Browser	Limitations
`IE 11 and older`	• Not officially supported.
`Edge (Version 78 and earlier)`	• Only image attachments can be downloaded by right clicking and choosing `save image as`. Downloading other types of attachments is not possible, since older versions of Edge do not fully support downloading data-urls. Everything else works well.
`Chrome/Chromium/` `Edge (Version 80+)`	• Attachments cannot be viewed or downloaded with a single click on it. As a workaround you can use the context menu and use `save as` or `open in another tab`.

Your browser must have Javascript enabled to view the test report. There are some plugins like `NoScript` that will likely cause problems. Also old versions of Internet Explorer do not load Javascript by default.

JUnit XML Test Report

HILSTER Testing Framework creates JUnit XML test reports that can be used in other tools like Continuous Integration systems, etc.

The JUnit XML format does not include all available data htf offers.

JSON Test Report

The JSON test report includes all test results with all data to be used in your tools.

YAML Test Report

The YAML test report includes all test results with all data to be used in your tools.

IBM® Engineering Requirements Management DOORS Test Report

The IBM® Engineering Requirements Management DOORS Test Report is a special test report that creates a DXL script to be imported into DOORS. Thus your test results can easily be imported to DOORS.

The DOORS Test Report works in conjunction with the `@meets_doors_requirements` decorator that attaches the tested requirements to individual tests.

```
@meets_doors_requirements("doors://host:port/path-without-id-",
                  "REQ_1", "REQ_2")
def test_with_useful_name():
    pass
```

The IBM Rational DOORS Test Report can only be used within Python code because it needs configuration parameters.

```
import htf

@htf.meets_doors_requirements("REQ_1", "REQ_2", "REQ_3", "TEST_1")
def test_with_doors_links():
    pass

htf.main(title="DOORS Test Report",
    extra_report=[
             DOORSTestReport(filename="doors.dxl",
                             report_module="/path/to/report/module",
                             requirements_modules={
                                 "REQ_.*" : "/path/to/requirements_module",
                                 "TEST_.*" : "/path/to/test_specification",
                             },
                             links_modules={
                                 "REQ_.*" : "/path/to/links/to/requirements_module",
                                 "TEST_.*" : "/path/to/links/to/test_specification",
                                 }
                             )
             ],
    )
```

```
class htf.DOORSTestReport(filename: str, report_module: str, requirements_modules: Dict[str,
str], links_modules: Dict[str, str] | None = None, render_meta_data: bool = True,
render_steps: bool = True)
```

`DOORSTestReport` creates a DOORS test report. Links are automatically created from the created DOORS module to the fulfilled requirements.

To state the fulfilled requirements the `meets_doors_requirements` decorator should be used.

Initialize a `DOORSTestReport` instance.

Parameters:

- **filename** – the filename for the DXL test report.
- **report_module** – the DOORS path to the report module.
- **requirements_modules** – maps links to DOORS modules. The keys are regular expressions and the values are the desired target modules.
- **links_modules={".*"** – “DOORS Links”}: maps requirements to desired link modules. The keys are regular expressions and the values are the desired link modules.
- **render_meta_data=True** – if set to *True* metadata is rendered in DOORS

- **render_steps=True** – if set to *True* steps are rendered in DOORS

```
render(data: Dict[str, Any]) → None
```

Generates and stores a DXL encoded TestReport.

Parameters:
data – test result data

Signed Reports

The HILSTER Testing Framework supports signed test reports since version `htf-1.2`.

Signed reports are a very useful feature to proof that a test report was not modified.

This could be useful when dealing with certification authorities or during legal proceedings to document when, how and to what degree a product was tested.

The signature is created using asymmetric cryptography. This allows for independent verification of the signatures validity with the public key.

Signed reports can be verified by others using the `checksignatures` command line utility which does not need to be licensed and which is shipped with `htf`.

Creating signed reports

Signed reports can only be created for the internal reports of the HILSTER Testing Framework. These are the HTML-report, the JUnit-xml-report and the JSON-Report. User supplied reports are not signed for security reasons.

To be able to create signed reports you need to license the `htf.signatures` feature. Once the feature is enabled `htf.signatures` signed reports are created automatically if you use `$ htf`, `$ htf run`, `$ htf dryrun`, `htf.main`, `htf.run` or `htf.dryrun`.

The signature is written into an extra file with the name of the report plus `.sig`.

To disable the creation of signatures you can set the environment variable `HTF_SIGN_REPORTS` to `"0"` using `set HTF_SIGN_REPORTS=0` on Windows or `export HTF_SIGN_REPORTS=0` on Linux.

Checking signatures

`checksignatures` is free to use and does not need a license.

To check signatures you can use the `checksignatures` commandline utility which is shipped with `htf`.

When called without any parameters `checksignatures` scans the current folder and prints the status of the files found.

Files without signatures can be omitted by using the `-s` option.

To verify specific files or folders, add them as parameters seperated by whitespace.

Report Tool

`report-tool` is a command line utility to convert reports into different formats, merge reports from different formats into different formats and extract attachments, stdout and stderr from reports to reduce their size.

Supported input formats are HTML, JUnit-XML, JSON and YAML.

Supported output formats are also HTML, JUnit-XML, JSON, YAML. Reports can also be uploaded to a Dashboard.

Usage

The usage is simple. To see all options run `report-tool -h`.

```
usage: report-tool [-h] [-t TITLE] [-H HTML_REPORT] [-x JUNIT_XML_REPORT] [-j JSON_REPORT] [-Y
YAML_REPORT] [-r REPORT_SERVER]
               [--extract-output] [--extract-attachments]
               report [report ...]

report-tool - HILSTER Testing Framework (HILSTER GmbH, https://hilster.io)

positional arguments:
  report                The input test report. All formats are supported. (may be repeated)

optional arguments:
  -h, --help            show this help message and exit
  -t TITLE, --title TITLE
                        Output testreport title
  -H HTML_REPORT, --html-report HTML_REPORT
                        Output HTML report filename (may be repeated)
  -x JUNIT_XML_REPORT, --junit-xml-report JUNIT_XML_REPORT
                        Output Junit-compatible XML report filename (may be repeated)
  -j JSON_REPORT, --json-report JSON_REPORT
                        Output JSON report filename (may be repeated)
  -Y YAML_REPORT, --yaml-report YAML_REPORT
                        Output YAML report filename (may be repeated)
  -r REPORT_SERVER, --report-server REPORT_SERVER
                        URL to report server (may be repeated)
  --extract-output
                        Extract STDOUT and STDERR from each test and store it into a text file.
  --extract-attachments
                        Extract attachments and store them locally.
```

Convert Reports

To convert a report into a different format add the source and add the outputs you want to create.

For example to create a JUnit-XML report from an HTML report run:

```
report-tool testreport.html --junit-xml-report testreport.xml
```

And to create all possible file formats run

```
report-tool testreport.html -x testreport.xml -H new_testreport.html -j testreport.json -Y
testreport.yml
```

Merge Reports

To merge reports add more than one source. All sources are combined automatically. The resulting start time is the minimum start time of all supplied sources. The resulting stop time is the maximum stop time of all supplied sources.

All test report metadata are put into every single test metadata with a prefix `'original'` and the test run metadata is replaced with the sources of the merge and the `report-tool` command used.

For example to create an HTML report from an HTML report and a JUnit-XML report run:

```
report-tool testreport.html results.xml -H merged.html
```

Upload Reports

To upload a report to your Dashboard use `-r` or `--report-server` with the url to your Dashboard and project.

In the following example `testreport.html` is uploaded to your Dashboard to project `example-project`.

```
report-tool testreport.html --report-server https://dashboard.lan/example-project
```

Of course you can also merge and upload reports with a single command. To merge `testreport.html` and `results.xml` and upload it run:

```
report-tool testreport.html results.xml --report-server https://dashboard.lan/example-project
```

Set the Report Title

To set the report title use `-t` or `--title`.

```
report-tool testreport.html --title "New Test Report" -H new_testreport.html
```

Signed Reports

If all input reports have a valid signature and the `signature` feature is enabled in your license a signature for the output will be created, too.

Extract Attachments

If your HTML report (or another report) contains attachments and is very big so your browser cannot open it anymore you can extract the attachments and write a new report without attachments. The extracted attachments are stored to disk with a reasonable filename and thus can be viewed easily.

To extract attachments from `testreport.html` and create `stripped.html` run:

```
report-tool testreport.html --extract-attachments --html-report stripped.html
```

Extract Output

If your HTML report (or another report) contains stdout and stderr and is very big so your browser cannot open it anymore you can extract stdout and stderr and write a new report without output. The extracted outputs are stored to disk with a reasonable filename and thus can be viewed easily.

To extract attachments from `testreport.html` and create `stripped.html` run:

```
report-tool testreport.html --extract-output --html-report stripped.html
```

Community Add-ons

The HILSTER Testing Framework community offers community add-ons in an external GitHub repository.

Modbus — The Modbus Testing Framework

`htf` includes a full-stack Modbus implementation since version 1.3 with the following features:

- Serial RTU, Serial ASCII, TCP, TLS and UDP support
- client support
- device simulator support
- default Modbus protocol
- extended Modbus protocol
- supported endianness: little endian, big endian, word aligned and half-word aligned
- protocol abstraction is based on `oser`
- working with data `oser` structures instead of bits and bytes
- fault insertion

• possibility to extend client and device simulator with own commands

The Modbus implementation is capable of testing Modbus clients and servers.

Examples

Simulating a Modbus-Device

```
import htf
from htf.modbus import ModbusTCPDevice, ModbusTCPClient


ADDRESS = 'localhost'
PORT = 10502  # must be able to run for non-root users


@htf.fixture(scope='class', name='device')
def create_device_simulator():
    device = ModbusTCPDevice(ADDRESS, port=PORT)
    yield device
    device.close()


@htf.fixture(scope='test', name='client')
def create_client():
    client = ModbusTCPClient(ADDRESS,
                             port=PORT,
                             references_start_at_one=False,
                             auto_disconnect=False,
                             debuglevel=0)  # set debuglevel to 1 or 2
    yield client
    client.close()


class ModbusTCPTests:
    """
    This tests shows how to simulate a modbus device and how to use the client to
    query it via TCP/IP.
    """

    def test_holding_registers(self, device, client, assertions):
        registers = client.read_multiple_holding_registers(0, 8)
        assertions.assert_equal(registers, [0] * 8)

        for address in range(10):
            client.write_multiple_holding_registers(address, [0, 1, 0, 1, 1, 0, 1, 0])
            registers = client.read_multiple_holding_registers(address, 8)
            assertions.assert_equal(registers, [0, 1, 0, 1, 1, 0, 1, 0])

            device.write_multiple_holding_registers(address, [1, 1, 0, 0, 1, 0, 1, 0])
            registers = client.read_multiple_holding_registers(address, 8)
            assertions.assert_equal(registers, [1, 1, 0, 0, 1, 0, 1, 0])

    def test_input_registers(self, device, client, assertions):
        device.write_multiple_input_registers(0, list(range(10)))

        registers = client.read_multiple_input_registers(0, 10)
```

```
        assertions.assert_equal(registers, list(range(10)))

        registers = device.read_multiple_input_registers(0, 10)
        assertions.assert_equal(registers, list(range(10)))

        device.write_multiple_input_registers(10, list(range(20, 31)))

        registers = client.read_multiple_input_registers(5, 15)
        assertions.assert_equal(registers, list(range(5, 10)) + list(range(20, 30)))

        for i in range(10):
            r = client.read_input_register(i)
            assertions.assert_equal(r, i)

            r = device.read_input_register(i)
            assertions.assert_equal(r, i)

        for i in range(10, 20):
            r = client.read_input_register(i)
            assertions.assert_equal(r, i+10)

            r = device.read_input_register(i)
            assertions.assert_equal(r, i + 10)

        device.write_input_register(2323, 1)
        r = client.read_input_register(2323)
        assertions.assert_equal(r, 1)


if __name__ == '__main__':
    htf.main()
```

Special Modbus Data Structures

```
import htf
import oser
from datetime import datetime
from htf.modbus import ModbusTCPDevice, ModbusTCPClient


ADDRESS = 'localhost'
PORT = 10502  # must be able to run for non-root users


class DateTime(oser.ByteStruct):
    def __init__(self, value=None):
        """
        ``DateTime`` data type.

        Args:
            value (datetime): a datetime instance to be used
        """
        super(DateTime, self).__init__()

        self.year = oser.UBInt16()
        self.month = oser.UBInt8()
        self.day = oser.UBInt8()
```

```
        self.hour = oser.UBInt8()
        self.minute = oser.UBInt8()
        self.second = oser.UBInt8()

        if value is not None:
            self.set(value)

    def set(self, value):
        """
        Set values from a ``datetime`` instance.

        Args:
            value (datetime): a ``datetime`` instance to be used
        """
        if not isinstance(value, datetime):
            raise ValueError("value must be of type datetime")

        self.year.set(value.year)
        self.month.set(value.month)
        self.day.set(value.day)

        self.hour.set(value.hour)
        self.minute.set(value.minute)
        self.second.set(value.second)

    def get(self):
        """
        Get values as datetime.

        Returns:
            datetime: an instance of ``datetime`` containing the current values
        """
        return datetime(year=self.year.get(), month=self.month.get(), day=self.day.get(),
                        hour=self.hour.get(), minute=self.minute.get(), second=self.second.get())


@htf.fixture(scope='class', name='device')
def create_device_simulator():
    device = ModbusTCPDevice(ADDRESS, port=PORT)
    yield device
    device.close()


@htf.fixture(scope='test', name='client')
def create_client():
    client = ModbusTCPClient(ADDRESS,
                             port=PORT,
                             references_start_at_one=False,
                             auto_disconnect=False,
                             debuglevel=0)  # set debuglevel to 1 or 2
    yield client
    client.close()


class ModbusDataStructureTests:
    """
    This test shows how to read and write own data structures using Modbus.
    """

    def test_datetime_data_structure(self, device, client, assertions):
```

```
        """
        A DateTime data type can be read from input registers at address 23.
        """
        now = datetime.now()
        device.write_input_register_data(23, DateTime(now))

        current_date_struct = client.read_input_register_data(23, DateTime)

        assertions.assert_equal(current_date_struct.year, now.year)
        assertions.assert_equal(current_date_struct.month, now.month)
        assertions.assert_equal(current_date_struct.day, now.day)

        assertions.assert_equal(current_date_struct.hour, now.hour)
        assertions.assert_equal(current_date_struct.minute, now.minute)
        assertions.assert_equal(current_date_struct.second, now.second)

        print("device date time is:", current_date_struct.get())


if __name__ == '__main__':
    htf.main()
```

Interfaces

Interfaces are used to implement communication. The user may add other interfaces if needed.

Serial RTU Client Interface

```
class htf.modbus.ModbusSerialRTUClientInterface(com_port: str | None = None, baud: int =
9600, bytesize: int = 8, parity: str = 'E', stopbits: int = 1, rtscts: bool = False, timeout:
int | float = 1.0, response_parser: ByteStruct | BitStruct | ByteType | BitType = <class
'htf.modbus.protocol.SerialRTUModbusResponse'>)
```

Parameters:

- **com_port** – the com port to be used
- **baud** – the baud rate
- **bytesize** – number of bits in a byte
- **parity** – serial.PARITY_NONE, serial.PARITY_ODD or serial.PARITY_EVEN
- **stopbits** – number of stopbits (1, 1.5 or 2)
- **rtscts** – if True use flow control
- **timeout** – the timeout in seconds
- **response_parser** – the parser to parse responses

```
close() → None
```

Close serial port.

```
read() → Tuple[bytes, None]
```

Read Modbus serial RTU responses from server.

Returns:
a tuple containing buffer, `None`

```
write(adu: SerialRTUModbusRequest) → None
```

Write `adu`.

Parameters:
adu – serializable RTU adu

Serial RTU Server Interface

```
class htf.modbus.ModbusSerialRTUServerInterface(com_port: str | None = None, baud: int =
9600, bytesize: int = 8, parity: str = 'E', stopbits: int = 1, rtscts: bool = False, timeout:
int | float = 1.0, response_parser: ByteStruct | BitStruct | ByteType | BitType = <class
'htf.modbus.protocol.SerialRTUModbusRequest'>)
```

Parameters:

- **com_port** – the com port to be used
- **baud** – the baud rate
- **bytesize** – number of bits in a byte
- **parity** – serial.PARITY_NONE, serial.PARITY_ODD or serial.PARITY_EVEN
- **stopbits** – number of stopbits (1, 1.5 or 2)
- **rtscts** – if True use flow control
- **timeout** – the timeout in seconds
- **response_parser** – the parser to parse responses

```
close() → None
```

Close serial port.

```
read() → Tuple[bytes, None]
```

Read Modbus serial RTU requests from client.

Returns:
a tuple containing buffer, `None`

```
write(adu: SerialRTUModbusRequest) → None
```

Write `adu`.

Parameters:
adu – serializable RTU adu

Serial ASCII Client Interface

```
class htf.modbus.ModbusSerialASCIIClientInterface(com_port: str | None = None, baud: int = 9600, bytesize: int = 8, parity: str = 'E', stopbits: int = 1, rtscts: bool = False, timeout: int | float = 1.0, response_parser: ByteStruct | BitStruct | ByteType | BitType = <class 'htf.modbus.protocol.SerialASCIIModbusResponse'>)
```

Parameters:

- **com_port** – the com port to be used
- **baud** – the baud rate
- **bytesize** – number of bits in a byte
- **parity** – serial.PARITY_NONE, serial.PARITY_ODD or serial.PARITY_EVEN
- **stopbits** – number of stopbits (1, 1.5 or 2)
- **rtscts** – if True use flow control
- **timeout** – the timeout in seconds
- **response_parser** – the parser to parse responses

```
change_ascii_delimiter(delimiter: bytes) → None
```

Change LF character to `delimiter`.

Parameters:
delimiter – bytes of length 1 containing the new ASCII delimiter

```
close() → None
```

Close serial port.

```
read() → Tuple[bytes, None]
```

Read Modbus serial ASCII responses from server. Read until line feed character.

Returns:
a tuple containing buffer, `None`

```
write(adu: SerialASCIIModbusRequest) → None
```

Write `adu`.

Parameters:
adu – serializable RTU adu

Serial ASCII Server Interface

```
class htf.modbus.ModbusSerialASCIIServerInterface(com_port: str | None = None, baud: int = 9600, bytesize: int = 8, parity: str = 'E', stopbits: int = 1, rtscts: bool = False, timeout: int | float = 1.0, response_parser: ByteStruct | BitStruct | ByteType | BitType = <class 'htf.modbus.protocol.SerialASCIIModbusRequest'>)
```

Parameters:

- **com_port** – the com port to be used
- **baud** – the baud rate
- **bytesize** – number of bits in a byte
- **parity** – serial.PARITY_NONE, serial.PARITY_ODD or serial.PARITY_EVEN
- **stopbits** – number of stopbits (1, 1.5 or 2)
- **rtscts** – if True use flow control
- **timeout** – the timeout in seconds
- **response_parser** – the parser to parse responses

```
change_ascii_delimiter(delimiter: bytes) → None
```

Change LF character to `delimiter`.

Parameters:
delimiter – bytes of length 1 containing the new ASCII delimiter

```
close() → None
```

Close serial port.

```
read() → Tuple[bytes, None]
```

Read serial ASCII Modbus requests from client.

Returns:
a tuple containing buffer, `None`

```
write(adu: SerialASCIIModbusRequest) → None
```

Write `adu`.

Parameters:
adu – serializable RTU adu

TCP Client Interface

```
class htf.modbus.ModbusTCPClientInterface(address: str, port: int = 502, timeout: int | float = 1.0, connect: bool = True)
```

Parameters:

- **address** – the ip-address to connect to
- **port** – the port
- **timeout** – the timeout in seconds
- **connect** – if set to `True` the socket connects automatically

```
close() → None
```

Close TCP connection.

```
open() → None
```

Open a socket.

```
read() → Tuple[bytes | None, None]
```

Read a TCP request from TCP server.

Returns:
a tuple containing buffer, `None`

```
reconnect() → None
```

Close socket and create a new one.

```
write(adu: TCPModbusRequest) → None
```

Write `adu`.

Parameters:

adu – serializable TCP adu

TCP Server Interface

```
class htf.modbus.ModbusTCPServerInterface(address: str, port: int = 502, timeout: int |
float = 1.0)
```

Parameters:

- **address** – the ip-address to listen on
- **port** – the port to listen on
- **timeout** – the timeout in seconds

```
accept() → Tuple[socket, Any]
```

Accept a new connection.

Returns:
a tuple containing connection, address

```
close() → None
```

Close TCP socket.

```
close_connection(connection: socket) → None
```

Close TCP connection.

```
read(connection: socket) → bytes
```

Read a TCP request from TLS client.

Returns:
a tuple containing buffer, `None`

```
write(connection: socket, adu: TCPModbusRequest) → None
```

Write `adu`.

Parameters:
adu – serializable TCP adu

UDP Client Interface

```
class htf.modbus.ModbusUDPClientInterface(address: str, port: int = 502, timeout: int |
float = 1.0)
```

Parameters:

- **address** – the ip-address to connect to
- **port** – the port
- **timeout** – the timeout in seconds

```
close() → None
```

Close TCP connection.

```
open() → None
```

Open a socket.

```
read() → Tuple[bytes | None, None]
```

Read a TLS request from TLS server.

Returns:
a tuple containing buffer, `None`

```
reconnect() → None
```

Close socket and create a new one.

```
write(adu: TCPModbusRequest) → None
```

Write `adu`.

Parameters:
adu – serializable TCP adu

UDP Server Interface

```
class htf.modbus.ModbusUDPServerInterface(address: str, port: int = 502, timeout:
Tuple[int, float] | None = None)
```

Parameters:

- **address** – the ip-address to listen on
- **port** – the port to listen on
- **timeout=None** – the timeout in seconds

```
close() → None
```

Close UDP socket.

```
read(bufsize: int = 300) → Tuple[bytes, Any]
```

Read a request from an UDP client.

Parameters:
bufsize – buffer size
Returns:
the received request

```
write(address: str, adu: TCPModbusRequest) → None
```

Write `adu`.

Parameters:

- **address** – the target ip-address
- **adu** – serializable TCP adu

TLS Client Interface

```
class htf.modbus.ModbusTLSClientInterface(address: str, port: int = 502, timeout: int |
float = 1.0, verify: bool = True, connect: bool = True, cafile: str | None = None, capath:
str | None = None, cadata: str | None = None)
```

Parameters:

- **address** – the ip-address to connect to
- **port** – the port
- **timeout** – the timeout in seconds
- **verify** – set to `False` to disable TLS certificate verification
- **connect** – if set to `True` the socket connects automatically
- **cafile** – path to a file of concatenated CA certificates in PEM format
- **capath** – path to a directory containing several CA certificates in PEM format
- **cadata** – either an ASCII string of one or more PEM-encoded certificates or a bytes-like object of DER-encoded certificates

```
close() → None
```

Close TCP connection.

```
open() → None
```

Open a socket.

```
read() → Tuple[bytes | None, None]
```

Read a TCP request from TCP server.

Returns:
a tuple containing buffer, `None`

```
reconnect() → None
```

Close socket and create a new one.

```
write(adu: TCPModbusRequest) → None
```

Write `adu`.

Parameters:
adu – serializable TCP adu

TLS Server Interface

```
class htf.modbus.ModbusTLSServerInterface(address: str, port: int = 502, timeout: int |
float = 1.0, certfile: str | None = None, keyfile: str | None = None, password: str | None =
None)
```

Parameters:

- **address** – the ip-address to listen on
- **port** – the port to listen on
- **timeout** – the timeout in seconds
- **certfile** – path to TLS certificate
- **keyfile** – path to TLS keyfile
- **password** – TLS password

```
accept() → Tuple[socket, Any]
```

Accept a new connection.

Returns:
a tuple containing connection, address

```
close() → None
```

Close TLS socket.

```
close_connection(connection: socket) → None
```

Close TCP connection.

```
read(connection: socket) → bytes
```

Read a TCP request from TLS client.

Returns:
a tuple containing buffer, `None`

```
write(connection: socket, adu: TCPModbusRequest) → None
```

Write `adu`.

Parameters:
adu – serializable TCP adu

Clients

Clients are used to query Modbus devices. Users can add own clients using own interfaces.

Base Client

```
class htf.modbus.ModbusBaseClient(interface: ModbusBaseInterface, request_parser:
Type[SerialRTUModbusRequest] | Type[SerialASCIIModbusRequest] | Type[TCPModbusRequest],
response_parser: Type[SerialRTUModbusResponse] | Type[SerialASCIIModbusResponse] |
Type[TCPModbusResponse], retries: int | None = None, references_start_at_one: bool = True,
debuglevel: int | None = None)
```

Parameters:

- **interface** – an interface to be used for communication
- **request_parser** – a parser for requests
- **response_parser** – a parser for responses
- **retries=None** – the number of retries until an exception is raised

- **references_start_at_one=True** – if set to `True` all references start at `1` else `0`
- **debuglevel** – the debuglevel (`None` disables debug, `1` prints requests and responses and `2` prints introspection)

```
build_request() → SerialRTUModbusRequest | SerialASCIIModbusRequest | TCPModbusRequest
```

Build and return an instance of the desired request parser.

Returns:
request parser

```
build_response() → SerialRTUModbusResponse | SerialASCIIModbusResponse |
TCPModbusResponse
```

Build and return an instance of the desired response parser.

Returns:
response parser

```
close() → None
```

Close the interface.

```
get_device_identification() → Dict[int, bytes]
```

Query and return the device identification using Modbus function code 43.14.

Returns:
containing key values pairs with the device identifiers

```
get_fifo_queue_content(address: int) → List[int]
```

Query and return the fifo content of `address` using Modbus function code 24.

Parameters:
address – the fifo queue pointer address
Returns:
list of content from the fifo queue at `address`

```
mask_holding_register(reference: int, and_mask: int, or_mask: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Mask write single holding register using Modbus function code 22.

Parameters:

- **reference** – the register reference
- **and_mask** – the and mask
- **or_mask** – the or mask

```
mask_multiple_holding_registers(reference: int, quantity: int, and_mask: int, or_mask: int) → None
```

Mask write multiple holding registers using Modbus function code 16.

Parameters:

- **reference** – the register reference
- **quantity** – the number of registers to be masked
- **and_mask** – the and mask
- **or_mask** – the or mask

```
query(adu: SerialRTUModbusRequest | SerialASCIIModbusRequest | TCPModbusRequest) → SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Query `adu` using the interface and return the decoded response. This method is thread-safe.

Parameters:
adu – a serializable adu
Returns:
the decoded response if no exception occurred
Return type:
adu
Raises:
ModbusException – in case of an exception response

```
read() → SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Read a message from the interface and decode it.

Returns:
the decoded adu as is
Return type:
adu
Raises:
Exception if no answer was received –

```
read_coil(reference: int) → int
```

Read a single coil using Modbus function code 1.

Parameters:

reference – the register reference.
Returns:
`1` if the coil is set and `0` else.

```
read_coil_bitfield(reference: int, length: int) → int
```

Read a bitfield from coil memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_coil_bitfield_list(reference: int, quantity: int, length: int) → List[int]
```

Read a list of bitfields from coil memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_coil_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct] |
Type[ByteType]) → ByteStruct | ByteType
```

Read data from coils using Modbus function code 1.

Parameters:

- **reference** – the coil reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_coil_flag(reference: int) → bool
```

Read a bit from coil memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_coil_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from coil memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_coil_nibble(reference: int) → int
```

Read a nibble from coil memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_coil_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from coil memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_discrete_input(reference: int) → int
```

Read single discrete input using Modbus function code 2.

Parameters:
reference – the register reference.
Returns:
`1` if the discrete input is set and `0` else.

```
read_discrete_input_bitfield(reference: int, length: int) → int
```

Read a bitfield from discrete input memory.

Parameters:

- **reference** – the reference to read from

- **length** – the length of the value

Returns:
int value read from `reference`

```
read_discrete_input_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_discrete_input_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct]
| Type[ByteType]) → ByteStruct | ByteType
```

Read data from discrete inputs using Modbus function code 2.

Parameters:

- **reference** – the discrete input reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_discrete_input_flag(reference: int) → bool
```

Read a bit from discrete input memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_discrete_input_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_discrete_input_nibble(reference: int) → int
```

Read a nibble from discrete input memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_discrete_input_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_extended_memory(memory_number: int, record_number: int, record_length: int) →
List[int]
```

Query and return extended memory from device using Modbus function code 20.

Parameters:

- **memory_number** – the memory number (0x0001 to 0xffff)
- **record_number** – the record to start reading from
- **record_length** – the length of records

Returns:
the read records

```
read_holding_register(reference: int) → int
```

Read a single holding register using Modbus function code 3.

Parameters:
reference – the register reference.
Returns:
the holding register value

```
read_holding_register_bdouble(reference: int) → float
```

Read a double from holding register memory.

Parameters:

reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_bdouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_bfloat(reference: int) → float
```

Read a float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_bfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_bitfield(reference: int, length: int) → int
```

Read a bitfield from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_holding_register_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → ByteStruct | ByteType
```

Read data from holding registers using Modbus function code 3.

Parameters:

- **reference** – the register reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_holding_register_double21436587(reference: int) → float
```

Read a 21436587-endian double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_double21436587_list(reference: int, quantity: int) → List[float]
```

Read a list of 21436587-endian doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_double78563412(reference: int) → float
```

Read a 78563412-endian double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_double78563412_list(reference: int, quantity: int) → List[float]
```

Read a list of 78563412-endian doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_flag(reference: int) → bool
```

Read a bit from holding register memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_holding_register_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_holding_register_float2143(reference: int) → float
```

Read a 2143-endian float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_float2143_list(reference: int, quantity: int) → List[float]
```

Read a list of 2143-endian floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_float3412(reference: int) → float
```

Read a 3412-endian float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_float3412_list(reference: int, quantity: int) → List[float]
```

Read a list of 3412-endian floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_ldouble(reference: int) → float
```

Read a double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_ldouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from holding register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_lfloat(reference: int) → float
```

Read a float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_lfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_nibble(reference: int) → int
```

Read a nibble from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint16(reference: int) → int
```

Read a signed 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint32(reference: int) → int
```

Read a signed 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint64(reference: int) → int
```

Read a signed 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint8(reference: int) → int
```

Read a signed 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint2143(reference: int) → int
```

Read a signed 2143-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint21436587(reference: int) → int
```

Read a signed 21436587-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:

int value read from `reference`

```
read_holding_register_sint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 21436587-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 2143-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint3412(reference: int) → int
```

Read a signed 3412-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 3412-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint78563412(reference: int) → int
```

Read a signed 78563412-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 78563412-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint16(reference: int) → int
```

Read a signed 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint32(reference: int) → int
```

Read a signed 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint64(reference: int) → int
```

Read a signed 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint8(reference: int) → int
```

Read a signed 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_string(reference: int, length: int) → bytes
```

Read a string from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
bytes value read from `reference`

```
read_holding_register_string_list(reference: int, quantity: int, length: int) →
List[bytes]
```

Read a list of strings from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of bytes values read from `reference`

```
read_holding_register_ubint16(reference: int) → int
```

Read an unsigned 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint32(reference: int) → int
```

Read an unsigned 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint64(reference: int) → int
```

Read an unsigned 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint8(reference: int) → int
```

Read an unsigned 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint2143(reference: int) → int
```

Read an unsigned 2143-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint21436587(reference: int) → int
```

Read an unsigned 21436587-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 21436587-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 2143-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint3412(reference: int) → int
```

Read an unsigned 3412-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 3412-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint78563412(reference: int) → int
```

Read an unsigned 78563412-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 78563412-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint16(reference: int) → int
```

Read an unsigned 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:

int value read from `reference`

```
read_holding_register_ulint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint32(reference: int) → int
```

Read an unsigned 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint64(reference: int) → int
```

Read an unsigned 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint8(reference: int) → int
```

Read an unsigned 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register(reference: int) → int
```

Read a single input register using Modbus function code 4.

Parameters:
reference – the register reference.
Returns:
the input register value

```
read_input_register_bdouble(reference: int) → float
```

Read a double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_bdouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_bfloat(reference: int) → float
```

Read a float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_bfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_bitfield(reference: int, length: int) → int
```

Read a bitfield from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_input_register_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct]
| Type[ByteType]) → ByteStruct | ByteType
```

Read data from input registers using Modbus function code 4.

Parameters:

- **reference** – the register reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_input_register_double21436587(reference: int) → float
```

Read a 21436587-endian double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_double21436587_list(reference: int, quantity: int) → List[float]
```

Read a list of 21436587-endian doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_double78563412(reference: int) → float
```

Read a 78563412-endian double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_double78563412_list(reference: int, quantity: int) → List[float]
```

Read a list of 78563412-endian doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_flag(reference: int) → bool
```

Read a bit from input register memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_input_register_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_input_register_float2143(reference: int) → float
```

Read a 2143-endian float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_float2143_list(reference: int, quantity: int) → List[float]
```

Read a list of 2143-endian floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_float3412(reference: int) → float
```

Read a 3412-endian float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_float3412_list(reference: int, quantity: int) → List[float]
```

Read a list of 3412-endian floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_ldouble(reference: int) → float
```

Read a double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_ldouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_lfloat(reference: int) → float
```

Read a float from input register memory.

Parameters:
reference – the reference to read from
Returns:

float value read from `reference`

```
read_input_register_lfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_nibble(reference: int) → int
```

Read a nibble from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint16(reference: int) → int
```

Read a signed 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint32(reference: int) → int
```

Read a signed 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint64(reference: int) → int
```

Read a signed 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint8(reference: int) → int
```

Read a signed 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint2143(reference: int) → int
```

Read a signed 2143-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint21436587(reference: int) → int
```

Read a signed 21436587-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 21436587-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:

list of int values read from `reference`

```
read_input_register_sint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 2143-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint3412(reference: int) → int
```

Read a signed 3412-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 3412-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint78563412(reference: int) → int
```

Read a signed 78563412-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 78563412-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint16(reference: int) → int
```

Read a signed 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint32(reference: int) → int
```

Read a signed 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint64(reference: int) → int
```

Read a signed 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint8(reference: int) → int
```

Read a signed 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_string(reference: int, length: int) → bytes
```

Read a string from input register memory.

Parameters:

- **reference** – the reference to read from

- **length** – the length of the value

Returns:
bytes value read from `reference`

```
read_input_register_string_list(reference: int, quantity: int, length: int) →
List[bytes]
```

Read a list of strings from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of bytes values read from `reference`

```
read_input_register_ubint16(reference: int) → int
```

Read an unsigned 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint32(reference: int) → int
```

Read an unsigned 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint64(reference: int) → int
```

Read an unsigned 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint8(reference: int) → int
```

Read an unsigned 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint2143(reference: int) → int
```

Read an unsigned 2143-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint21436587(reference: int) → int
```

Read an unsigned 21436587-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 21436587-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 2143-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint3412(reference: int) → int
```

Read an unsigned 3412-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 3412-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint78563412(reference: int) → int
```

Read an unsigned 78563412-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 78563412-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint16(reference: int) → int
```

Read an unsigned 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint32(reference: int) → int
```

Read an unsigned 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint64(reference: int) → int
```

Read an unsigned 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint8(reference: int) → int
```

Read an unsigned 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_multiple_coils(reference: int, quantity: int) → List[int]
```

Read multiple coils using Modbus function code 1.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of coils to be read.

Returns:
a list of coil states where each element is `1` if the corresponding coil is set and `0` else.

```
read_multiple_discrete_inputs(reference: int, quantity: int) → List[int]
```

Read multiple discrete inputs using Modbus function code 2.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of discrete inputs to be read.

Returns:
a list of discrete inputs where each element is `1` if the corresponding discrete input is set and `0` else.

```
read_multiple_holding_registers(reference: int, quantity: int) → List[int]
```

Read multiple holding registers using Modbus function code 3.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of registers to be read.

Returns:
the holding register values

```
read_multiple_input_registers(reference: int, quantity: int) → List[int]
```

Read multiple input registers using Modbus function code 4.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of registers to be read.

Returns:
the input register values

```
read_write_multiple_holding_registers(read_reference: int, read_quantity: int,
write_reference: int, write_values: List[int]) → List[int]
```

Read/Write multiple holding registers using Modbus function code 23.

Parameters:

- **read_reference** – the register reference to read from.
- **read_quantity** – the number of registers to read.
- **write_reference** – the register reference to to write to.
- **write_values** – the values to be written.

Returns:
the read registers

```
write(adu: SerialRTUModbusRequest | SerialASCIIModbusRequest | TCPModbusRequest) → None
```

Write `adu` encoded using the interface.

Parameters:
adu – a serializable adu

```
write_coil(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a single coil using Modbus function code 5.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written.

```
write_coil_bitfield(reference: int, value: int, length: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_coil_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_coil_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct] |
Type[ByteType]) → SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write data to coils using Modbus function code 15.

Parameters:

- **reference** – the coil reference.
- **data** – the type or instance or the decoder for the data

```
write_coil_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_coil_flag_list(reference: int, values: List[bool]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_coil_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_coil_nibble_list(reference: int, values: List[int]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_extended_memory(memory_number: int, record_number: int, records: List[int]) → None
```

Write extended memory using Modbus function code 21.

Parameters:

- **memory_number** – the memory number (0x0001 to 0xffff)
- **record_number** – the record to write to
- **records** – the records to be written

```
write_holding_register(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write single holding register using Modbus function code 6.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written into the corresponding register.

```
write_holding_register_bdouble(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_bdouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_bfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_bfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_bitfield(reference: int, value: int, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

- **length** – the length of the value

```
write_holding_register_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_holding_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → SerialRTUModbusResponse | SerialASCIIModbusResponse |
TCPModbusResponse
```

Write data to holding registers using Modbus function code 16.

Parameters:

- **reference** – the register reference.
- **data** – the type or instance or the decoder for the data

```
write_holding_register_double21436587(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 21436587-endian double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_double21436587_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 21436587-endian doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_double78563412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 78563412-endian double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_double78563412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 78563412-endian doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_holding_register_flag_list(reference: int, values: List[bool]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_holding_register_float2143(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 2143-endian float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_float2143_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 2143-endian floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_float3412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 3412-endian float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_float3412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 3412-endian floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_ldouble(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_ldouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_lfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_lfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_nibble_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 2143-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 21436587-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 21436587-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 2143-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 3412-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 3412-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 78563412-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 78563412-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_string(reference: int, value: bytes, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a string to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bytes value to be written
- **length** – the length of the value

```
write_holding_register_string_list(reference: int, values: List[bytes], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of strings to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bytes values to be written
- **length** – the length of the value

```
write_holding_register_ubint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 2143-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 21436587-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 21436587-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 2143-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 3412-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 3412-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 78563412-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 78563412-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_multiple_coils(reference: int, values: List[int]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write multiple coils using Modbus function code 15.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

```
write_multiple_holding_registers(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write multiple holding registers using Modbus function code 16.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

Serial RTU Client

```
class htf.modbus.ModbusSerialRTUClient(com_port: str, baud: int = 9600, bytesize: int = 8,
parity: str = 'E', stopbits: int = 1, rtscts: bool = False, device_address: int = 1, timeout:
int | float = 1.0, retries: int | None = None, references_start_at_one: bool = True,
request_parser: Type[SerialRTUModbusRequest] | Type[SerialASCIIModbusRequest] |
Type[TCPModbusRequest] = <class 'htf.modbus.protocol.SerialRTUModbusRequest'>,
response_parser: Type[SerialRTUModbusResponse] | Type[SerialASCIIModbusResponse] |
Type[TCPModbusResponse] = <class 'htf.modbus.protocol.SerialRTUModbusResponse'>, debuglevel:
int | None = None)
```

Parameters:

- **com_port** – the com port to be used
- **baud** – the baud rate
- **bytesize** – number of bits in a byte
- **parity** – serial.PARITY_NONE, serial.PARITY_ODD or serial.PARITY_EVEN
- **stopbits** – number of stopbits (1, 1.5 or 2)
- **rtscts** – if True use flow control
- **device_address** – the device address (Slave ID)
- **timeout** – the timeout in seconds
- **retries=None** – the number of retries until an exception is raised
- **references_start_at_one=True** – if set to `True` all references start at `1` else `0`
- **request_parser** – a parser for requests
- **response_parser** – a parser for responses
- **debuglevel** – the debuglevel (`None` disables debug, `1` prints requests and responses and `2` prints introspection)

```
build_request() → SerialRTUModbusRequest | SerialASCIIModbusRequest | TCPModbusRequest
```

Build and return an instance of the desired request parser.

Returns:
request parser

```
build_response() → SerialRTUModbusResponse | SerialASCIIModbusResponse |
TCPModbusResponse
```

Build and return an instance of the desired response parser.

Returns:
response parser

```
clear_counters_and_diagnostic_register() → None
```

Clear counters and diagnostic register using Modbus function code 8.10.

```
clear_overrun_counter_and_flag() → None
```

Clear overrun counter and reset the error flag using Modbus function code 8.20.

```
close() → None
```

Close the interface.

```
disable_broken_checksums() → None
```

Disable broken checksums for testing purposes.

```
echo(data: bytes) → SerialRTUModbusResponse | SerialASCIIModbusResponse |
TCPModbusResponse
```

Query data and return echo using Modbus function code 8.0. An exception is raised if the received data differ from the sent data or the request is too long.

Parameters:
data – The bytes to be sent.
Returns:
response

```
enable_broken_checksums() → None
```

Enable broken checksums for testing purposes.

```
get_bus_character_overrun_count() → int
```

Query and return the bus character overrun count from the device using Modbus function code 8.18.

Returns:
the bus character overrun count

```
get_bus_communication_error_count() → int
```

Query and return the bus communication error count from the device using Modbus function code 8.12.

Returns:
the bus communication error count

```
get_bus_exception_error_count() → int
```

Query and return the bus exception error count from the device using Modbus function code 8.13.

Returns:
the bus exception error count

```
get_bus_message_count() → int
```

Query and return the bus message count from the device using Modbus function code 8.11.

Returns:
the bus message count

```
get_busy_count() → int
```

Query and return the slave busy count from the device using Modbus function code 8.17.

Returns:
the slave busy count

```
get_busy_status() → bool
```

Query and return busy status using Modbus function code 11.

Returns:
busy_status

```
get_communication_event_counter_and_busy_status() → Tuple[int, bool]
```

Query and return communication event counter and busy status using Modbus function code 11.

Returns:
a tuple containing communication_event_counter (int), busy_status (bool)

```
get_device_identification() → Dict[int, bytes]
```

Query and return the device identification using Modbus function code 43.14.

Returns:
containing key values pairs with the device identifiers

```
get_diagnostic_register() → bytes
```

Query and return the diagnostic register using Modbus function 8.2.

Returns:
the content of the diagnostic register

```
get_event_count() → int
```

Query and return event count status using Modbus function code 11.

Returns:
communication_event_counter

```
get_event_log() → Tuple[int, int, int, List[int]]
```

Query and return the event log using Modbus function code 12.

Returns:
a tuple containing busy_status (int), event_count (int), message_count (int), events (list of int)

```
get_exception_status() → int
```

Read and return exception status using Modbus function code 7.

Returns:
the current exception status

```
get_fifo_queue_content(address: int) → List[int]
```

Query and return the fifo content of `address` using Modbus function code 24.

Parameters:
address – the fifo queue pointer address
Returns:
list of content from the fifo queue at `address`

```
get_slave_id() → Tuple[int, bool, bytes]
```

Query and return the slave id using Modbus function code 17.

Returns:
a tuple containing slave-id, running, data

```
get_slave_message_count() → int
```

Query and return the slave message count from the device using Modbus function code 8.14.

Returns:
the slave message count

```
get_slave_nak_count() → int
```

Query and return the slave nak count from the device using Modbus function code 8.16.

Returns:
the slave nak count

```
get_slave_no_response_count() → int
```

Query and return the slave no response count from the device using Modbus function code 8.15.

Returns:
the slave no response count

```
mask_holding_register(reference: int, and_mask: int, or_mask: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Mask write single holding register using Modbus function code 22.

Parameters:

- **reference** – the register reference
- **and_mask** – the and mask
- **or_mask** – the or mask

```
mask_multiple_holding_registers(reference: int, quantity: int, and_mask: int, or_mask: int) → None
```

Mask write multiple holding registers using Modbus function code 16.

Parameters:

- **reference** – the register reference
- **quantity** – the number of registers to be masked
- **and_mask** – the and mask
- **or_mask** – the or mask

```
query(adu: SerialRTUModbusRequest | SerialASCIIModbusRequest | TCPModbusRequest) → SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Query `adu` using the interface and return the decoded response. This method is thread-safe.

Parameters:
adu – a serializable adu
Returns:
the decoded response if no exception occurred
Return type:
adu
Raises:
ModbusException – in case of an exception response

```
read() → SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Read a message from the interface and decode it.

Returns:
the decoded adu as is
Return type:
adu
Raises:
Exception if no answer was received –

```
read_coil(reference: int) → int
```

Read a single coil using Modbus function code 1.

Parameters:
reference – the register reference.
Returns:
`1` if the coil is set and `0` else.

```
read_coil_bitfield(reference: int, length: int) → int
```

Read a bitfield from coil memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_coil_bitfield_list(reference: int, quantity: int, length: int) → List[int]
```

Read a list of bitfields from coil memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_coil_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct] |
Type[ByteType]) → ByteStruct | ByteType
```

Read data from coils using Modbus function code 1.

Parameters:

- **reference** – the coil reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_coil_flag(reference: int) → bool
```

Read a bit from coil memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_coil_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from coil memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_coil_nibble(reference: int) → int
```

Read a nibble from coil memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_coil_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from coil memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_discrete_input(reference: int) → int
```

Read single discrete input using Modbus function code 2.

Parameters:
reference – the register reference.
Returns:
`1` if the discrete input is set and `0` else.

```
read_discrete_input_bitfield(reference: int, length: int) → int
```

Read a bitfield from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_discrete_input_bitfield_list(reference: int, quantity: int, length: int) → List[int]
```

Read a list of bitfields from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_discrete_input_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct] | Type[ByteType]) → ByteStruct | ByteType
```

Read data from discrete inputs using Modbus function code 2.

Parameters:

- **reference** – the discrete input reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_discrete_input_flag(reference: int) → bool
```

Read a bit from discrete input memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_discrete_input_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_discrete_input_nibble(reference: int) → int
```

Read a nibble from discrete input memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_discrete_input_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_extended_memory(memory_number: int, record_number: int, record_length: int) →
List[int]
```

Query and return extended memory from device using Modbus function code 20.

Parameters:

- **memory_number** – the memory number (0x0001 to 0xffff)
- **record_number** – the record to start reading from
- **record_length** – the length of records

Returns:
the read records

```
read_holding_register(reference: int) → int
```

Read a single holding register using Modbus function code 3.

Parameters:
reference – the register reference.
Returns:
the holding register value

```
read_holding_register_bdouble(reference: int) → float
```

Read a double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_bdouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_bfloat(reference: int) → float
```

Read a float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_bfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_bitfield(reference: int, length: int) → int
```

Read a bitfield from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_holding_register_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → ByteStruct | ByteType
```

Read data from holding registers using Modbus function code 3.

Parameters:

- **reference** – the register reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_holding_register_double21436587(reference: int) → float
```

Read a 21436587-endian double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_double21436587_list(reference: int, quantity: int) → List[float]
```

Read a list of 21436587-endian doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_double78563412(reference: int) → float
```

Read a 78563412-endian double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_double78563412_list(reference: int, quantity: int) → List[float]
```

Read a list of 78563412-endian doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_flag(reference: int) → bool
```

Read a bit from holding register memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_holding_register_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_holding_register_float2143(reference: int) → float
```

Read a 2143-endian float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_float2143_list(reference: int, quantity: int) → List[float]
```

Read a list of 2143-endian floats from holding register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_float3412(reference: int) → float
```

Read a 3412-endian float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_float3412_list(reference: int, quantity: int) → List[float]
```

Read a list of 3412-endian floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_ldouble(reference: int) → float
```

Read a double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_ldouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_lfloat(reference: int) → float
```

Read a float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_lfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_nibble(reference: int) → int
```

Read a nibble from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint16(reference: int) → int
```

Read a signed 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint32(reference: int) → int
```

Read a signed 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint64(reference: int) → int
```

Read a signed 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint8(reference: int) → int
```

Read a signed 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint2143(reference: int) → int
```

Read a signed 2143-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint21436587(reference: int) → int
```

Read a signed 21436587-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 21436587-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 2143-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint3412(reference: int) → int
```

Read a signed 3412-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 3412-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint78563412(reference: int) → int
```

Read a signed 78563412-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 78563412-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint16(reference: int) → int
```

Read a signed 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint32(reference: int) → int
```

Read a signed 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint64(reference: int) → int
```

Read a signed 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint8(reference: int) → int
```

Read a signed 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_string(reference: int, length: int) → bytes
```

Read a string from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
bytes value read from `reference`

```
read_holding_register_string_list(reference: int, quantity: int, length: int) →
List[bytes]
```

Read a list of strings from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of bytes values read from `reference`

```
read_holding_register_ubint16(reference: int) → int
```

Read an unsigned 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint32(reference: int) → int
```

Read an unsigned 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from

Returns:
int value read from `reference`

```
read_holding_register_ubint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint64(reference: int) → int
```

Read an unsigned 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint8(reference: int) → int
```

Read an unsigned 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint2143(reference: int) → int
```

Read an unsigned 2143-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint21436587(reference: int) → int
```

Read an unsigned 21436587-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 21436587-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 2143-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint3412(reference: int) → int
```

Read an unsigned 3412-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 3412-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint78563412(reference: int) → int
```

Read an unsigned 78563412-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 78563412-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint16(reference: int) → int
```

Read an unsigned 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:

int value read from `reference`

```
read_holding_register_ulint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint32(reference: int) → int
```

Read an unsigned 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint64(reference: int) → int
```

Read an unsigned 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint8(reference: int) → int
```

Read an unsigned 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register(reference: int) → int
```

Read a single input register using Modbus function code 4.

Parameters:
reference – the register reference.
Returns:
the input register value

```
read_input_register_bdouble(reference: int) → float
```

Read a double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_bdouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_bfloat(reference: int) → float
```

Read a float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_bfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_bitfield(reference: int, length: int) → int
```

Read a bitfield from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_input_register_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct]
| Type[ByteType]) → ByteStruct | ByteType
```

Read data from input registers using Modbus function code 4.

Parameters:

- **reference** – the register reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_input_register_double21436587(reference: int) → float
```

Read a 21436587-endian double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_double21436587_list(reference: int, quantity: int) → List[float]
```

Read a list of 21436587-endian doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_double78563412(reference: int) → float
```

Read a 78563412-endian double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_double78563412_list(reference: int, quantity: int) → List[float]
```

Read a list of 78563412-endian doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_flag(reference: int) → bool
```

Read a bit from input register memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_input_register_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_input_register_float2143(reference: int) → float
```

Read a 2143-endian float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_float2143_list(reference: int, quantity: int) → List[float]
```

Read a list of 2143-endian floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_float3412(reference: int) → float
```

Read a 3412-endian float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_float3412_list(reference: int, quantity: int) → List[float]
```

Read a list of 3412-endian floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_ldouble(reference: int) → float
```

Read a double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_ldouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_lfloat(reference: int) → float
```

Read a float from input register memory.

Parameters:
reference – the reference to read from
Returns:

float value read from `reference`

```
read_input_register_lfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_nibble(reference: int) → int
```

Read a nibble from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint16(reference: int) → int
```

Read a signed 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint32(reference: int) → int
```

Read a signed 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint64(reference: int) → int
```

Read a signed 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint8(reference: int) → int
```

Read a signed 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint2143(reference: int) → int
```

Read a signed 2143-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint21436587(reference: int) → int
```

Read a signed 21436587-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 21436587-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:

list of int values read from `reference`

```
read_input_register_sint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 2143-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint3412(reference: int) → int
```

Read a signed 3412-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 3412-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint78563412(reference: int) → int
```

Read a signed 78563412-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 78563412-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint16(reference: int) → int
```

Read a signed 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint32(reference: int) → int
```

Read a signed 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint64(reference: int) → int
```

Read a signed 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint8(reference: int) → int
```

Read a signed 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_string(reference: int, length: int) → bytes
```

Read a string from input register memory.

Parameters:

- **reference** – the reference to read from

- **length** – the length of the value

Returns:
bytes value read from `reference`

```
read_input_register_string_list(reference: int, quantity: int, length: int) →
List[bytes]
```

Read a list of strings from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of bytes values read from `reference`

```
read_input_register_ubint16(reference: int) → int
```

Read an unsigned 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint32(reference: int) → int
```

Read an unsigned 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint64(reference: int) → int
```

Read an unsigned 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint8(reference: int) → int
```

Read an unsigned 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint2143(reference: int) → int
```

Read an unsigned 2143-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint21436587(reference: int) → int
```

Read an unsigned 21436587-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 21436587-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 2143-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint3412(reference: int) → int
```

Read an unsigned 3412-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 3412-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint78563412(reference: int) → int
```

Read an unsigned 78563412-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 78563412-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint16(reference: int) → int
```

Read an unsigned 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint32(reference: int) → int
```

Read an unsigned 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint64(reference: int) → int
```

Read an unsigned 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint8(reference: int) → int
```

Read an unsigned 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_multiple_coils(reference: int, quantity: int) → List[int]
```

Read multiple coils using Modbus function code 1.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of coils to be read.

Returns:
a list of coil states where each element is `1` if the corresponding coil is set and `0` else.

```
read_multiple_discrete_inputs(reference: int, quantity: int) → List[int]
```

Read multiple discrete inputs using Modbus function code 2.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of discrete inputs to be read.

Returns:
a list of discrete inputs where each element is `1` if the corresponding discrete input is set and `0` else.

```
read_multiple_holding_registers(reference: int, quantity: int) → List[int]
```

Read multiple holding registers using Modbus function code 3.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of registers to be read.

Returns:
the holding register values

```
read_multiple_input_registers(reference: int, quantity: int) → List[int]
```

Read multiple input registers using Modbus function code 4.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of registers to be read.

Returns:
the input register values

```
read_write_multiple_holding_registers(read_reference: int, read_quantity: int,
write_reference: int, write_values: List[int]) → List[int]
```

Read/Write multiple holding registers using Modbus function code 23.

Parameters:

- **read_reference** – the register reference to read from.
- **read_quantity** – the number of registers to read.
- **write_reference** – the register reference to to write to.
- **write_values** – the values to be written.

Returns:
the read registers

```
restart_communications(reset_eventlog: bool = False) → None
```

Restart communications using Modbus function code 8.1.

```
set_device_address(device_address: int) → None
```

Set device address.

Parameters:
device_address – the device address to be used for the next requests.

```
set_listen_only_mode() → None
```

Set listen only mode in server using Modbus function code 8.4.

```
write(adu: SerialRTUModbusRequest | SerialASCIIModbusRequest | TCPModbusRequest) → None
```

Write `adu` encoded using the interface.

Parameters:
adu – a serializable adu

```
write_coil(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a single coil using Modbus function code 5.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written.

```
write_coil_bitfield(reference: int, value: int, length: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_coil_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_coil_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct] |
Type[ByteType]) → SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write data to coils using Modbus function code 15.

Parameters:

- **reference** – the coil reference.
- **data** – the type or instance or the decoder for the data

```
write_coil_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_coil_flag_list(reference: int, values: List[bool]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_coil_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_coil_nibble_list(reference: int, values: List[int]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_extended_memory(memory_number: int, record_number: int, records: List[int]) → None
```

Write extended memory using Modbus function code 21.

Parameters:

- **memory_number** – the memory number (0x0001 to 0xffff)
- **record_number** – the record to write to
- **records** – the records to be written

```
write_holding_register(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write single holding register using Modbus function code 6.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written into the corresponding register.

```
write_holding_register_bdouble(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_bdouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_bfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_bfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_bitfield(reference: int, value: int, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_holding_register_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_holding_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → SerialRTUModbusResponse | SerialASCIIModbusResponse |
TCPModbusResponse
```

Write data to holding registers using Modbus function code 16.

Parameters:

- **reference** – the register reference.
- **data** – the type or instance or the decoder for the data

```
write_holding_register_double21436587(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 21436587-endian double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_double21436587_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 21436587-endian doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_double78563412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 78563412-endian double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_double78563412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 78563412-endian doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_holding_register_flag_list(reference: int, values: List[bool]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_holding_register_float2143(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 2143-endian float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_float2143_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 2143-endian floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_float3412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 3412-endian float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_float3412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 3412-endian floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_ldouble(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_ldouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_lfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_lfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_nibble_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 2143-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 21436587-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 21436587-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 2143-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 3412-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 3412-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 78563412-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 78563412-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_string(reference: int, value: bytes, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a string to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bytes value to be written
- **length** – the length of the value

```
write_holding_register_string_list(reference: int, values: List[bytes], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of strings to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bytes values to be written
- **length** – the length of the value

```
write_holding_register_ubint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 2143-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 21436587-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 21436587-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 2143-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 3412-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 3412-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 78563412-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 78563412-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_multiple_coils(reference: int, values: List[int]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write multiple coils using Modbus function code 15.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

```
write_multiple_holding_registers(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write multiple holding registers using Modbus function code 16.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

Serial ASCII Client

```
class htf.modbus.ModbusSerialASCIIClient(com_port: str, baud: int = 9600,
bytesize: int = 8, parity: str = 'E', stopbits: int = 1, rtscts: bool = False,
device_address: int = 1, timeout: int | float = 1.0, retries: int | None = None,
references_start_at_one: bool = True, ascii_delimiter: bytes = b'\n', request_parser:
Type[SerialRTUModbusRequest] | Type[SerialASCIIModbusRequest] | Type[TCPModbusRequest] =
<class 'htf.modbus.protocol.SerialASCIIModbusRequest'>, response_parser:
Type[SerialRTUModbusResponse] | Type[SerialASCIIModbusResponse] | Type[TCPModbusResponse] =
<class 'htf.modbus.protocol.SerialASCIIModbusResponse'>, debuglevel: int | None = None)
```

Parameters:

- **com_port** – the com port to be used
- **baud** – the baud rate
- **bytesize** – number of bits in a byte
- **parity** – serial.PARITY_NONE, serial.PARITY_ODD or serial.PARITY_EVEN
- **stopbits** – number of stopbits (1, 1.5 or 2)
- **rtscts** – if True use flow control

- **device_address** – the device address (Slave ID)
- **timeout** – the timeout in seconds
- **retries=None** – the number of retries until an exception is raised
- **references_start_at_one=True** – if set to `True` all references start at `1` else `0`
- **ascii_delimiter** – the ascii delimiter to be used
- **request_parser** – a parser for requests
- **response_parser** – a parser for responses
- **debuglevel** – the debuglevel (`None` disables debug, `1` prints requests and responses and `2` prints introspection)

```
build_request() → SerialRTUModbusRequest | SerialASCIIModbusRequest | TCPModbusRequest
```

Build and return an instance of the desired request parser.

Returns:
request parser

```
build_response() → SerialRTUModbusResponse | SerialASCIIModbusResponse |
TCPModbusResponse
```

Build and return an instance of the desired response parser.

Returns:
response parser

```
change_ascii_delimiter(delimiter: bytes) → None
```

Change ASCII delimiter in the device.

Parameters:
delimiter – the new ASCII delimiter.

```
clear_counters_and_diagnostic_register() → None
```

Clear counters and diagnostic register using Modbus function code 8.10.

```
clear_overrun_counter_and_flag() → None
```

Clear overrun counter and reset the error flag using Modbus function code 8.20.

```
close() → None
```

Close the interface.

```
disable_broken_checksums() → None
```

Disable broken checksums for testing purposes.

```
echo(data: bytes) → SerialRTUModbusResponse | SerialASCIIModbusResponse |
TCPModbusResponse
```

Query data and return echo using Modbus function code 8.0. An exception is raised if the received data differ from the sent data or the request is too long.

Parameters:
data – The bytes to be sent.
Returns:
response

```
enable_broken_checksums() → None
```

Enable broken checksums for testing purposes.

```
get_bus_character_overrun_count() → int
```

Query and return the bus character overrun count from the device using Modbus function code 8.18.

Returns:
the bus character overrun count

```
get_bus_communication_error_count() → int
```

Query and return the bus communication error count from the device using Modbus function code 8.12.

Returns:
the bus communication error count

```
get_bus_exception_error_count() → int
```

Query and return the bus exception error count from the device using Modbus function code 8.13.

Returns:
the bus exception error count

```
get_bus_message_count() → int
```

Query and return the bus message count from the device using Modbus function code 8.11.

Returns:
the bus message count

```
get_busy_count() → int
```

Query and return the slave busy count from the device using Modbus function code 8.17.

Returns:
the slave busy count

```
get_busy_status() → bool
```

Query and return busy status using Modbus function code 11.

Returns:
busy_status

```
get_communication_event_counter_and_busy_status() → Tuple[int, bool]
```

Query and return communication event counter and busy status using Modbus function code 11.

Returns:
a tuple containing communication_event_counter (int), busy_status (bool)

```
get_device_identification() → Dict[int, bytes]
```

Query and return the device identification using Modbus function code 43.14.

Returns:
containing key values pairs with the device identifiers

```
get_diagnostic_register() → bytes
```

Query and return the diagnostic register using Modbus function 8.2.

Returns:
the content of the diagnostic register

```
get_event_count() → int
```

Query and return event count status using Modbus function code 11.

Returns:
communication_event_counter

```
get_event_log() → Tuple[int, int, int, List[int]]
```

Query and return the event log using Modbus function code 12.

Returns:
a tuple containing busy_status (int), event_count (int), message_count (int), events (list of int)

```
get_exception_status() → int
```

Read and return exception status using Modbus function code 7.

Returns:
the current exception status

```
get_fifo_queue_content(address: int) → List[int]
```

Query and return the fifo content of `address` using Modbus function code 24.

Parameters:
address – the fifo queue pointer address
Returns:
list of content from the fifo queue at `address`

```
get_slave_id() → Tuple[int, bool, bytes]
```

Query and return the slave id using Modbus function code 17.

Returns:
a tuple containing slave-id, running, data

```
get_slave_message_count() → int
```

Query and return the slave message count from the device using Modbus function code 8.14.

Returns:
the slave message count

```
get_slave_nak_count() → int
```

Query and return the slave nak count from the device using Modbus function code 8.16.

Returns:
the slave nak count

```
get_slave_no_response_count() → int
```

Query and return the slave no response count from the device using Modbus function code 8.15.

Returns:
the slave no response count

```
mask_holding_register(reference: int, and_mask: int, or_mask: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Mask write single holding register using Modbus function code 22.

Parameters:

- **reference** – the register reference
- **and_mask** – the and mask
- **or_mask** – the or mask

```
mask_multiple_holding_registers(reference: int, quantity: int, and_mask: int, or_mask:
int) → None
```

Mask write multiple holding registers using Modbus function code 16.

Parameters:

- **reference** – the register reference
- **quantity** – the number of registers to be masked
- **and_mask** – the and mask
- **or_mask** – the or mask

```
query(adu: SerialRTUModbusRequest | SerialASCIIModbusRequest | TCPModbusRequest) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Query `adu` using the interface and return the decoded response. This method is thread-safe.

Parameters:
adu – a serializable adu
Returns:
the decoded response if no exception occurred

Return type:
adu
Raises:
ModbusException – in case of an exception response

```
read() → SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Read a message from the interface and decode it.

Returns:
the decoded adu as is
Return type:
adu
Raises:
Exception if no answer was received –

```
read_coil(reference: int) → int
```

Read a single coil using Modbus function code 1.

Parameters:
reference – the register reference.
Returns:
`1` if the coil is set and `0` else.

```
read_coil_bitfield(reference: int, length: int) → int
```

Read a bitfield from coil memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_coil_bitfield_list(reference: int, quantity: int, length: int) → List[int]
```

Read a list of bitfields from coil memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_coil_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct] |
Type[ByteType]) → ByteStruct | ByteType
```

Read data from coils using Modbus function code 1.

Parameters:

- **reference** – the coil reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_coil_flag(reference: int) → bool
```

Read a bit from coil memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_coil_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from coil memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_coil_nibble(reference: int) → int
```

Read a nibble from coil memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_coil_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from coil memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_discrete_input(reference: int) → int
```

Read single discrete input using Modbus function code 2.

Parameters:
reference – the register reference.
Returns:
`1` if the discrete input is set and `0` else.

```
read_discrete_input_bitfield(reference: int, length: int) → int
```

Read a bitfield from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_discrete_input_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_discrete_input_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct]
| Type[ByteType]) → ByteStruct | ByteType
```

Read data from discrete inputs using Modbus function code 2.

Parameters:

- **reference** – the discrete input reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_discrete_input_flag(reference: int) → bool
```

Read a bit from discrete input memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_discrete_input_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_discrete_input_nibble(reference: int) → int
```

Read a nibble from discrete input memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_discrete_input_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_extended_memory(memory_number: int, record_number: int, record_length: int) →
List[int]
```

Query and return extended memory from device using Modbus function code 20.

Parameters:

- **memory_number** – the memory number (0x0001 to 0xffff)

- **record_number** – the record to start reading from
- **record_length** – the length of records

Returns:
the read records

```
read_holding_register(reference: int) → int
```

Read a single holding register using Modbus function code 3.

Parameters:
reference – the register reference.
Returns:
the holding register value

```
read_holding_register_bdouble(reference: int) → float
```

Read a double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_bdouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_bfloat(reference: int) → float
```

Read a float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_bfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_bitfield(reference: int, length: int) → int
```

Read a bitfield from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_holding_register_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → ByteStruct | ByteType
```

Read data from holding registers using Modbus function code 3.

Parameters:

- **reference** – the register reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_holding_register_double21436587(reference: int) → float
```

Read a 21436587-endian double from holding register memory.

Parameters:

reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_double21436587_list(reference: int, quantity: int) → List[float]
```

Read a list of 21436587-endian doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_double78563412(reference: int) → float
```

Read a 78563412-endian double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_double78563412_list(reference: int, quantity: int) → List[float]
```

Read a list of 78563412-endian doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_flag(reference: int) → bool
```

Read a bit from holding register memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_holding_register_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_holding_register_float2143(reference: int) → float
```

Read a 2143-endian float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_float2143_list(reference: int, quantity: int) → List[float]
```

Read a list of 2143-endian floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_float3412(reference: int) → float
```

Read a 3412-endian float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_float3412_list(reference: int, quantity: int) → List[float]
```

Read a list of 3412-endian floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_ldouble(reference: int) → float
```

Read a double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_ldouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_lfloat(reference: int) → float
```

Read a float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_lfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_nibble(reference: int) → int
```

Read a nibble from holding register memory.

Parameters:
reference – the reference to read from
Returns:

int value read from `reference`

```
read_holding_register_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint16(reference: int) → int
```

Read a signed 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint32(reference: int) → int
```

Read a signed 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint64(reference: int) → int
```

Read a signed 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint8(reference: int) → int
```

Read a signed 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint2143(reference: int) → int
```

Read a signed 2143-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint21436587(reference: int) → int
```

Read a signed 21436587-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 21436587-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 2143-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint3412(reference: int) → int
```

Read a signed 3412-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:

int value read from `reference`

```
read_holding_register_sint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 3412-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint78563412(reference: int) → int
```

Read a signed 78563412-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 78563412-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint16(reference: int) → int
```

Read a signed 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint32(reference: int) → int
```

Read a signed 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint64(reference: int) → int
```

Read a signed 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint8(reference: int) → int
```

Read a signed 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_string(reference: int, length: int) → bytes
```

Read a string from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
bytes value read from `reference`

```
read_holding_register_string_list(reference: int, quantity: int, length: int) →
List[bytes]
```

Read a list of strings from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of bytes values read from `reference`

```
read_holding_register_ubint16(reference: int) → int
```

Read an unsigned 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint32(reference: int) → int
```

Read an unsigned 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint64(reference: int) → int
```

Read an unsigned 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint8(reference: int) → int
```

Read an unsigned 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint2143(reference: int) → int
```

Read an unsigned 2143-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint21436587(reference: int) → int
```

Read an unsigned 21436587-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 21436587-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 2143-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint3412(reference: int) → int
```

Read an unsigned 3412-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 3412-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint78563412(reference: int) → int
```

Read an unsigned 78563412-endian 64-bit integer from holding register memory.

Parameters:

reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 78563412-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint16(reference: int) → int
```

Read an unsigned 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint32(reference: int) → int
```

Read an unsigned 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint64(reference: int) → int
```

Read an unsigned 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint8(reference: int) → int
```

Read an unsigned 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register(reference: int) → int
```

Read a single input register using Modbus function code 4.

Parameters:
reference – the register reference.
Returns:
the input register value

```
read_input_register_bdouble(reference: int) → float
```

Read a double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_bdouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_bfloat(reference: int) → float
```

Read a float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_bfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:

list of float values read from `reference`

```
read_input_register_bitfield(reference: int, length: int) → int
```

Read a bitfield from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_input_register_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct]
| Type[ByteType]) → ByteStruct | ByteType
```

Read data from input registers using Modbus function code 4.

Parameters:

- **reference** – the register reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_input_register_double21436587(reference: int) → float
```

Read a 21436587-endian double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_double21436587_list(reference: int, quantity: int) → List[float]
```

Read a list of 21436587-endian doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_double78563412(reference: int) → float
```

Read a 78563412-endian double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_double78563412_list(reference: int, quantity: int) → List[float]
```

Read a list of 78563412-endian doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_flag(reference: int) → bool
```

Read a bit from input register memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_input_register_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from input register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_input_register_float2143(reference: int) → float
```

Read a 2143-endian float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_float2143_list(reference: int, quantity: int) → List[float]
```

Read a list of 2143-endian floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_float3412(reference: int) → float
```

Read a 3412-endian float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_float3412_list(reference: int, quantity: int) → List[float]
```

Read a list of 3412-endian floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_ldouble(reference: int) → float
```

Read a double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_ldouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_lfloat(reference: int) → float
```

Read a float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_lfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_nibble(reference: int) → int
```

Read a nibble from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint16(reference: int) → int
```

Read a signed 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint32(reference: int) → int
```

Read a signed 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint64(reference: int) → int
```

Read a signed 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint8(reference: int) → int
```

Read a signed 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint2143(reference: int) → int
```

Read a signed 2143-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint21436587(reference: int) → int
```

Read a signed 21436587-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 21436587-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 2143-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint3412(reference: int) → int
```

Read a signed 3412-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 3412-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint78563412(reference: int) → int
```

Read a signed 78563412-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 78563412-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint16(reference: int) → int
```

Read a signed 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint32(reference: int) → int
```

Read a signed 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint64(reference: int) → int
```

Read a signed 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint8(reference: int) → int
```

Read a signed 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_string(reference: int, length: int) → bytes
```

Read a string from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
bytes value read from `reference`

```
read_input_register_string_list(reference: int, quantity: int, length: int) →
List[bytes]
```

Read a list of strings from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of bytes values read from `reference`

```
read_input_register_ubint16(reference: int) → int
```

Read an unsigned 16-bit integer from input register memory.

Parameters:
reference – the reference to read from

Returns:
int value read from `reference`

```
read_input_register_ubint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint32(reference: int) → int
```

Read an unsigned 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint64(reference: int) → int
```

Read an unsigned 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint8(reference: int) → int
```

Read an unsigned 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint2143(reference: int) → int
```

Read an unsigned 2143-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint21436587(reference: int) → int
```

Read an unsigned 21436587-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 21436587-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 2143-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint3412(reference: int) → int
```

Read an unsigned 3412-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 3412-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint78563412(reference: int) → int
```

Read an unsigned 78563412-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 78563412-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint16(reference: int) → int
```

Read an unsigned 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint32(reference: int) → int
```

Read an unsigned 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint64(reference: int) → int
```

Read an unsigned 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint8(reference: int) → int
```

Read an unsigned 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_multiple_coils(reference: int, quantity: int) → List[int]
```

Read multiple coils using Modbus function code 1.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of coils to be read.

Returns:
a list of coil states where each element is `1` if the corresponding coil is set and `0` else.

```
read_multiple_discrete_inputs(reference: int, quantity: int) → List[int]
```

Read multiple discrete inputs using Modbus function code 2.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of discrete inputs to be read.

Returns:
a list of discrete inputs where each element is `1` if the corresponding discrete input is set and `0` else.

```
read_multiple_holding_registers(reference: int, quantity: int) → List[int]
```

Read multiple holding registers using Modbus function code 3.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of registers to be read.

Returns:
the holding register values

```
read_multiple_input_registers(reference: int, quantity: int) → List[int]
```

Read multiple input registers using Modbus function code 4.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of registers to be read.

Returns:
the input register values

```
read_write_multiple_holding_registers(read_reference: int, read_quantity: int,
write_reference: int, write_values: List[int]) → List[int]
```

Read/Write multiple holding registers using Modbus function code 23.

Parameters:

- **read_reference** – the register reference to read from.
- **read_quantity** – the number of registers to read.
- **write_reference** – the register reference to to write to.
- **write_values** – the values to be written.

Returns:
the read registers

```
restart_communications(reset_eventlog: bool = False) → None
```

Restart communications using Modbus function code 8.1.

```
set_device_address(device_address: int) → None
```

Set device address.

Parameters:
device_address – the device address to be used for the next requests.

```
set_listen_only_mode() → None
```

Set listen only mode in server using Modbus function code 8.4.

```
write(adu: SerialRTUModbusRequest | SerialASCIIModbusRequest | TCPModbusRequest) → None
```

Write `adu` encoded using the interface.

Parameters:
adu – a serializable adu

```
write_coil(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a single coil using Modbus function code 5.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written.

```
write_coil_bitfield(reference: int, value: int, length: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_coil_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_coil_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct] |
Type[ByteType]) → SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write data to coils using Modbus function code 15.

Parameters:

- **reference** – the coil reference.
- **data** – the type or instance or the decoder for the data

```
write_coil_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_coil_flag_list(reference: int, values: List[bool]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_coil_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_coil_nibble_list(reference: int, values: List[int]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_extended_memory(memory_number: int, record_number: int, records: List[int]) → None
```

Write extended memory using Modbus function code 21.

Parameters:

- **memory_number** – the memory number (0x0001 to 0xffff)
- **record_number** – the record to write to
- **records** – the records to be written

```
write_holding_register(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write single holding register using Modbus function code 6.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written into the corresponding register.

```
write_holding_register_bdouble(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_bdouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_bfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_bfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_bitfield(reference: int, value: int, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_holding_register_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to holding register memory.

Parameters:

- **reference** – the reference to write to

- **values** – a list of int values to be written
- **length** – the length of the value

```
write_holding_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → SerialRTUModbusResponse | SerialASCIIModbusResponse |
TCPModbusResponse
```

Write data to holding registers using Modbus function code 16.

Parameters:

- **reference** – the register reference.
- **data** – the type or instance or the decoder for the data

```
write_holding_register_double21436587(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 21436587-endian double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_double21436587_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 21436587-endian doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_double78563412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 78563412-endian double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_double78563412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 78563412-endian doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_holding_register_flag_list(reference: int, values: List[bool]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_holding_register_float2143(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 2143-endian float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_float2143_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 2143-endian floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_float3412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 3412-endian float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_float3412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 3412-endian floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_ldouble(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_ldouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_lfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_lfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_nibble_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 2143-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 21436587-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 21436587-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 2143-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 3412-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 3412-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 78563412-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 78563412-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_string(reference: int, value: bytes, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a string to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bytes value to be written
- **length** – the length of the value

```
write_holding_register_string_list(reference: int, values: List[bytes], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of strings to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bytes values to be written
- **length** – the length of the value

```
write_holding_register_ubint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 2143-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 21436587-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 21436587-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 2143-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 3412-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 3412-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 78563412-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 78563412-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_multiple_coils(reference: int, values: List[int]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write multiple coils using Modbus function code 15.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

```
write_multiple_holding_registers(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write multiple holding registers using Modbus function code 16.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

TCP Client

```
class htf.modbus.ModbusTCPClient(address: str, port: int = 502, unit_identifier: int = 0,
timeout: int | float = 1.0, retries: int | None = None, references_start_at_one: bool = True,
transaction_identifier: int = 0, auto_disconnect: bool = False, request_parser:
Type[SerialRTUModbusRequest] | Type[SerialASCIIModbusRequest] | Type[TCPModbusRequest] =
<class 'htf.modbus.protocol.TCPModbusRequest'>, response_parser:
Type[SerialRTUModbusResponse] | Type[SerialASCIIModbusResponse] | Type[TCPModbusResponse] =
<class 'htf.modbus.protocol.TCPModbusResponse'>, debuglevel: int | None = None)
```

Parameters:

- **address** – the ip address to connect to
- **port** – the port to be used
- **unit_identifier** – the unit identifier for Modbus TCP
- **timeout** – the timeout in seconds
- **retries=None** – the number of retries until an exception is raised
- **references_start_at_one=True** – if set to `True` all references start at `1` else `0`
- **transaction_identifier** – the initial transaction identifier
- **auto_disconnect=False** – if set to `True` the TCP connection is closed immediately and re-opened on next request
- **request_parser** – a parser for requests
- **response_parser** – a parser for responses
- **debuglevel** – the debuglevel (`None` disables debug, `1` prints requests and responses and `2` prints introspection)

```
build_request() → SerialRTUModbusRequest | SerialASCIIModbusRequest | TCPModbusRequest
```

Build and return an instance of the desired request parser.

Returns:
request parser

```
build_response() → SerialRTUModbusResponse | SerialASCIIModbusResponse |
TCPModbusResponse
```

Build and return an instance of the desired response parser.

Returns:
response parser

```
close() → None
```

Close the interface.

```
get_device_identification() → Dict[int, bytes]
```

Query and return the device identification using Modbus function code 43.14.

Returns:
containing key values pairs with the device identifiers

```
get_fifo_queue_content(address: int) → List[int]
```

Query and return the fifo content of `address` using Modbus function code 24.

Parameters:
address – the fifo queue pointer address
Returns:
list of content from the fifo queue at `address`

```
get_transaction_identifier() → int
```

Create and return a new transaction identifer. Is thread-safe.

Returns:
the next transaction identifier

```
mask_holding_register(reference: int, and_mask: int, or_mask: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Mask write single holding register using Modbus function code 22.

Parameters:

- **reference** – the register reference
- **and_mask** – the and mask
- **or_mask** – the or mask

```
mask_multiple_holding_registers(reference: int, quantity: int, and_mask: int, or_mask:
int) → None
```

Mask write multiple holding registers using Modbus function code 16.

Parameters:

- **reference** – the register reference
- **quantity** – the number of registers to be masked
- **and_mask** – the and mask
- **or_mask** – the or mask

```
query(adu: SerialRTUModbusRequest | SerialASCIIModbusRequest | TCPModbusRequest) → SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Query `adu` using the interface and return the decoded response.

Parameters:
adu – a serializable adu
Returns:
the decoded response if no exception occurred
Return type:
adu
Raises:
ModbusException – in case of an exception response

```
read() → SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Read a message from the interface and decode it.

Returns:
the decoded adu as is
Return type:
adu
Raises:
Exception if no answer was received –

```
read_coil(reference: int) → int
```

Read a single coil using Modbus function code 1.

Parameters:
reference – the register reference.
Returns:
`1` if the coil is set and `0` else.

```
read_coil_bitfield(reference: int, length: int) → int
```

Read a bitfield from coil memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_coil_bitfield_list(reference: int, quantity: int, length: int) → List[int]
```

Read a list of bitfields from coil memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_coil_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct] |
Type[ByteType]) → ByteStruct | ByteType
```

Read data from coils using Modbus function code 1.

Parameters:

- **reference** – the coil reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_coil_flag(reference: int) → bool
```

Read a bit from coil memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_coil_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from coil memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_coil_nibble(reference: int) → int
```

Read a nibble from coil memory.

Parameters:
reference – the reference to read from

Returns:
int value read from `reference`

```
read_coil_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from coil memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_discrete_input(reference: int) → int
```

Read single discrete input using Modbus function code 2.

Parameters:
reference – the register reference.
Returns:
`1` if the discrete input is set and `0` else.

```
read_discrete_input_bitfield(reference: int, length: int) → int
```

Read a bitfield from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_discrete_input_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_discrete_input_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct]
| Type[ByteType]) → ByteStruct | ByteType
```

Read data from discrete inputs using Modbus function code 2.

Parameters:

- **reference** – the discrete input reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_discrete_input_flag(reference: int) → bool
```

Read a bit from discrete input memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_discrete_input_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_discrete_input_nibble(reference: int) → int
```

Read a nibble from discrete input memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_discrete_input_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from discrete input memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_extended_memory(memory_number: int, record_number: int, record_length: int) →
List[int]
```

Query and return extended memory from device using Modbus function code 20.

Parameters:

- **memory_number** – the memory number (0x0001 to 0xffff)
- **record_number** – the record to start reading from
- **record_length** – the length of records

Returns:
the read records

```
read_holding_register(reference: int) → int
```

Read a single holding register using Modbus function code 3.

Parameters:
reference – the register reference.
Returns:
the holding register value

```
read_holding_register_bdouble(reference: int) → float
```

Read a double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_bdouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_bfloat(reference: int) → float
```

Read a float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_bfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_bitfield(reference: int, length: int) → int
```

Read a bitfield from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_holding_register_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → ByteStruct | ByteType
```

Read data from holding registers using Modbus function code 3.

Parameters:

- **reference** – the register reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_holding_register_double21436587(reference: int) → float
```

Read a 21436587-endian double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_double21436587_list(reference: int, quantity: int) → List[float]
```

Read a list of 21436587-endian doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_double78563412(reference: int) → float
```

Read a 78563412-endian double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_double78563412_list(reference: int, quantity: int) → List[float]
```

Read a list of 78563412-endian doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_flag(reference: int) → bool
```

Read a bit from holding register memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_holding_register_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_holding_register_float2143(reference: int) → float
```

Read a 2143-endian float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_float2143_list(reference: int, quantity: int) → List[float]
```

Read a list of 2143-endian floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_float3412(reference: int) → float
```

Read a 3412-endian float from holding register memory.

Parameters:
reference – the reference to read from
Returns:

float value read from `reference`

```
read_holding_register_float3412_list(reference: int, quantity: int) → List[float]
```

Read a list of 3412-endian floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_ldouble(reference: int) → float
```

Read a double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_ldouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_lfloat(reference: int) → float
```

Read a float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_lfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_nibble(reference: int) → int
```

Read a nibble from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint16(reference: int) → int
```

Read a signed 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint32(reference: int) → int
```

Read a signed 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint64(reference: int) → int
```

Read a signed 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint8(reference: int) → int
```

Read a signed 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:

int value read from `reference`

```
read_holding_register_sbint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint2143(reference: int) → int
```

Read a signed 2143-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint21436587(reference: int) → int
```

Read a signed 21436587-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 21436587-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 2143-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint3412(reference: int) → int
```

Read a signed 3412-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 3412-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint78563412(reference: int) → int
```

Read a signed 78563412-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 78563412-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint16(reference: int) → int
```

Read a signed 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint32(reference: int) → int
```

Read a signed 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint64(reference: int) → int
```

Read a signed 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:

int value read from `reference`

```
read_holding_register_slint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint8(reference: int) → int
```

Read a signed 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_string(reference: int, length: int) → bytes
```

Read a string from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
bytes value read from `reference`

```
read_holding_register_string_list(reference: int, quantity: int, length: int) →
List[bytes]
```

Read a list of strings from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of bytes values read from `reference`

```
read_holding_register_ubint16(reference: int) → int
```

Read an unsigned 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint32(reference: int) → int
```

Read an unsigned 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:

list of int values read from `reference`

```
read_holding_register_ubint64(reference: int) → int
```

Read an unsigned 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint8(reference: int) → int
```

Read an unsigned 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint2143(reference: int) → int
```

Read an unsigned 2143-endian 32-bit integer from holding register memory.

Parameters:

reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint21436587(reference: int) → int
```

Read an unsigned 21436587-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 21436587-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 2143-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint3412(reference: int) → int
```

Read an unsigned 3412-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 3412-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint78563412(reference: int) → int
```

Read an unsigned 78563412-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 78563412-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint16(reference: int) → int
```

Read an unsigned 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint32(reference: int) → int
```

Read an unsigned 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint64(reference: int) → int
```

Read an unsigned 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint8(reference: int) → int
```

Read an unsigned 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:

int value read from `reference`

```
read_holding_register_ulint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register(reference: int) → int
```

Read a single input register using Modbus function code 4.

Parameters:
reference – the register reference.
Returns:
the input register value

```
read_input_register_bdouble(reference: int) → float
```

Read a double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_bdouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_bfloat(reference: int) → float
```

Read a float from input register memory.

Parameters:

reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_bfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_bitfield(reference: int, length: int) → int
```

Read a bitfield from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_input_register_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct]
| Type[ByteType]) → ByteStruct | ByteType
```

Read data from input registers using Modbus function code 4.

Parameters:

- **reference** – the register reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_input_register_double21436587(reference: int) → float
```

Read a 21436587-endian double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_double21436587_list(reference: int, quantity: int) → List[float]
```

Read a list of 21436587-endian doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_double78563412(reference: int) → float
```

Read a 78563412-endian double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_double78563412_list(reference: int, quantity: int) → List[float]
```

Read a list of 78563412-endian doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_flag(reference: int) → bool
```

Read a bit from input register memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_input_register_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_input_register_float2143(reference: int) → float
```

Read a 2143-endian float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_float2143_list(reference: int, quantity: int) → List[float]
```

Read a list of 2143-endian floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_float3412(reference: int) → float
```

Read a 3412-endian float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_float3412_list(reference: int, quantity: int) → List[float]
```

Read a list of 3412-endian floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_ldouble(reference: int) → float
```

Read a double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_ldouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_lfloat(reference: int) → float
```

Read a float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_lfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_nibble(reference: int) → int
```

Read a nibble from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint16(reference: int) → int
```

Read a signed 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint32(reference: int) → int
```

Read a signed 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:

int value read from `reference`

```
read_input_register_sbint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint64(reference: int) → int
```

Read a signed 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint8(reference: int) → int
```

Read a signed 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint2143(reference: int) → int
```

Read a signed 2143-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint21436587(reference: int) → int
```

Read a signed 21436587-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 21436587-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 2143-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint3412(reference: int) → int
```

Read a signed 3412-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 3412-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint78563412(reference: int) → int
```

Read a signed 78563412-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 78563412-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint16(reference: int) → int
```

Read a signed 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:

int value read from `reference`

```
read_input_register_slint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint32(reference: int) → int
```

Read a signed 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint64(reference: int) → int
```

Read a signed 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint8(reference: int) → int
```

Read a signed 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_string(reference: int, length: int) → bytes
```

Read a string from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
bytes value read from `reference`

```
read_input_register_string_list(reference: int, quantity: int, length: int) →
List[bytes]
```

Read a list of strings from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of bytes values read from `reference`

```
read_input_register_ubint16(reference: int) → int
```

Read an unsigned 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint32(reference: int) → int
```

Read an unsigned 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint64(reference: int) → int
```

Read an unsigned 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint8(reference: int) → int
```

Read an unsigned 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint2143(reference: int) → int
```

Read an unsigned 2143-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint21436587(reference: int) → int
```

Read an unsigned 21436587-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 21436587-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 2143-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint3412(reference: int) → int
```

Read an unsigned 3412-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 3412-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint78563412(reference: int) → int
```

Read an unsigned 78563412-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 78563412-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint16(reference: int) → int
```

Read an unsigned 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint32(reference: int) → int
```

Read an unsigned 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:

int value read from `reference`

```
read_input_register_ulint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint64(reference: int) → int
```

Read an unsigned 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint8(reference: int) → int
```

Read an unsigned 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_multiple_coils(reference: int, quantity: int) → List[int]
```

Read multiple coils using Modbus function code 1.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of coils to be read.

Returns:
a list of coil states where each element is `1` if the corresponding coil is set and `0` else.

```
read_multiple_discrete_inputs(reference: int, quantity: int) → List[int]
```

Read multiple discrete inputs using Modbus function code 2.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of discrete inputs to be read.

Returns:
a list of discrete inputs where each element is `1` if the corresponding discrete input is set and `0` else.

```
read_multiple_holding_registers(reference: int, quantity: int) → List[int]
```

Read multiple holding registers using Modbus function code 3.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of registers to be read.

Returns:
the holding register values

```
read_multiple_input_registers(reference: int, quantity: int) → List[int]
```

Read multiple input registers using Modbus function code 4.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of registers to be read.

Returns:

the input register values

```
read_write_multiple_holding_registers(read_reference: int, read_quantity: int,
write_reference: int, write_values: List[int]) → List[int]
```

Read/Write multiple holding registers using Modbus function code 23.

Parameters:

- **read_reference** – the register reference to read from.
- **read_quantity** – the number of registers to read.
- **write_reference** – the register reference to to write to.
- **write_values** – the values to be written.

Returns:
the read registers

```
reconnect() → None
```

Reconnect socket.

```
write(adu: SerialRTUModbusRequest | SerialASCIIModbusRequest | TCPModbusRequest,
auto_length: bool = True) → None
```

Write `adu` encoded using the interface. The transaction identifier is set automatically. The unit identifier is put into the adu automatically, too.

Parameters:

- **adu** – a serializable adu
- **auto_length=True** – if set to `True` the length is calculated automatically

```
write_coil(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a single coil using Modbus function code 5.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written.

```
write_coil_bitfield(reference: int, value: int, length: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_coil_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_coil_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct] |
Type[ByteType]) → SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write data to coils using Modbus function code 15.

Parameters:

- **reference** – the coil reference.
- **data** – the type or instance or the decoder for the data

```
write_coil_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_coil_flag_list(reference: int, values: List[bool]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_coil_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_coil_nibble_list(reference: int, values: List[int]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_extended_memory(memory_number: int, record_number: int, records: List[int]) → None
```

Write extended memory using Modbus function code 21.

Parameters:

- **memory_number** – the memory number (0x0001 to 0xffff)
- **record_number** – the record to write to
- **records** – the records to be written

```
write_holding_register(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write single holding register using Modbus function code 6.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written into the corresponding register.

```
write_holding_register_bdouble(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_bdouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_bfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_bfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_bitfield(reference: int, value: int, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_holding_register_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to holding register memory.

Parameters:

- **reference** – the reference to write to

- **values** – a list of int values to be written
- **length** – the length of the value

```
write_holding_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → SerialRTUModbusResponse | SerialASCIIModbusResponse |
TCPModbusResponse
```

Write data to holding registers using Modbus function code 16.

Parameters:

- **reference** – the register reference.
- **data** – the type or instance or the decoder for the data

```
write_holding_register_double21436587(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 21436587-endian double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_double21436587_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 21436587-endian doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_double78563412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 78563412-endian double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_double78563412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 78563412-endian doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_holding_register_flag_list(reference: int, values: List[bool]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_holding_register_float2143(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 2143-endian float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_float2143_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 2143-endian floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_float3412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 3412-endian float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_float3412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 3412-endian floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_ldouble(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_ldouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_lfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_lfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_nibble_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 2143-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 21436587-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 21436587-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 2143-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 3412-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 3412-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 78563412-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 78563412-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_string(reference: int, value: bytes, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a string to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bytes value to be written
- **length** – the length of the value

```
write_holding_register_string_list(reference: int, values: List[bytes], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of strings to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bytes values to be written
- **length** – the length of the value

```
write_holding_register_ubint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 2143-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 21436587-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 21436587-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 2143-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 3412-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 3412-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 78563412-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 78563412-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_multiple_coils(reference: int, values: List[int]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write multiple coils using Modbus function code 15.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

```
write_multiple_holding_registers(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write multiple holding registers using Modbus function code 16.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

TLS Client

```
class htf.modbus.ModbusTLSClient(address: str, port: int = 502, unit_identifier: int = 0,
timeout: int | float = 1.0, retries: int | None = None, references_start_at_one: bool = True,
transaction_identifier: int = 0, auto_disconnect: bool = False, verify: bool = True, cafile:
str | None = None, capath: str | None = None, cadata: str | None = None, request_parser:
Type[SerialRTUModbusRequest] | Type[SerialASCIIModbusRequest] | Type[TCPModbusRequest] =
<class 'htf.modbus.protocol.TCPModbusRequest'>, response_parser:
Type[SerialRTUModbusResponse] | Type[SerialASCIIModbusResponse] | Type[TCPModbusResponse] =
<class 'htf.modbus.protocol.TCPModbusResponse'>, debuglevel: int | None = None)
```

Parameters:

- **address** – the ip address to connect to
- **port** – the port to be used
- **unit_identifier** – the unit identifier for Modbus TCP
- **timeout** – the timeout in seconds
- **retries=None** – the number of retries until an exception is raised
- **references_start_at_one=True** – if set to `True` all references start at `1` else `0`
- **transaction_identifier** – the initial transaction identifier
- **auto_disconnect=False** – if set to `True` the TCP connection is closed immediately and re-opened on next request
- **cafile** – path to a file of concatenated CA certificates in PEM format
- **capath** – path to a directory containing several CA certificates in PEM format
- **cadata** – either an ASCII string of one or more PEM-encoded certificates or a bytes-like object of DER-encoded certificates
- **request_parser** – a parser for requests
- **response_parser** – a parser for responses
- **debuglevel** – the debuglevel (`None` disables debug, `1` prints requests and responses and `2` prints introspection)

```
build_request() → SerialRTUModbusRequest | SerialASCIIModbusRequest | TCPModbusRequest
```

Build and return an instance of the desired request parser.

Returns:
request parser

```
build_response() → SerialRTUModbusResponse | SerialASCIIModbusResponse |
TCPModbusResponse
```

Build and return an instance of the desired response parser.

Returns:
response parser

```
close() → None
```

Close the interface.

```
get_device_identification() → Dict[int, bytes]
```

Query and return the device identification using Modbus function code 43.14.

Returns:
containing key values pairs with the device identifiers

```
get_fifo_queue_content(address: int) → List[int]
```

Query and return the fifo content of `address` using Modbus function code 24.

Parameters:
address – the fifo queue pointer address
Returns:
list of content from the fifo queue at `address`

```
get_transaction_identifier() → int
```

Create and return a new transaction identifer. Is thread-safe.

Returns:
the next transaction identifier

```
mask_holding_register(reference: int, and_mask: int, or_mask: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Mask write single holding register using Modbus function code 22.

Parameters:

- **reference** – the register reference
- **and_mask** – the and mask
- **or_mask** – the or mask

```
mask_multiple_holding_registers(reference: int, quantity: int, and_mask: int, or_mask:
int) → None
```

Mask write multiple holding registers using Modbus function code 16.

Parameters:

- **reference** – the register reference

- **quantity** – the number of registers to be masked
- **and_mask** – the and mask
- **or_mask** – the or mask

```
query(adu: SerialRTUModbusRequest | SerialASCIIModbusRequest | TCPModbusRequest) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Query `adu` using the interface and return the decoded response.

Parameters:
adu – a serializable adu
Returns:
the decoded response if no exception occurred
Return type:
adu
Raises:
ModbusException – in case of an exception response

```
read() → SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Read a message from the interface and decode it.

Returns:
the decoded adu as is
Return type:
adu
Raises:
Exception if no answer was received –

```
read_coil(reference: int) → int
```

Read a single coil using Modbus function code 1.

Parameters:
reference – the register reference.
Returns:
`1` if the coil is set and `0` else.

```
read_coil_bitfield(reference: int, length: int) → int
```

Read a bitfield from coil memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:

int value read from `reference`

```
read_coil_bitfield_list(reference: int, quantity: int, length: int) → List[int]
```

Read a list of bitfields from coil memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_coil_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct] |
Type[ByteType]) → ByteStruct | ByteType
```

Read data from coils using Modbus function code 1.

Parameters:

- **reference** – the coil reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_coil_flag(reference: int) → bool
```

Read a bit from coil memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_coil_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from coil memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_coil_nibble(reference: int) → int
```

Read a nibble from coil memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_coil_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from coil memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_discrete_input(reference: int) → int
```

Read single discrete input using Modbus function code 2.

Parameters:
reference – the register reference.
Returns:
`1` if the discrete input is set and `0` else.

```
read_discrete_input_bitfield(reference: int, length: int) → int
```

Read a bitfield from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_discrete_input_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from discrete input memory.

Parameters:

- **reference** – the reference to read from

- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_discrete_input_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct]
| Type[ByteType]) → ByteStruct | ByteType
```

Read data from discrete inputs using Modbus function code 2.

Parameters:

- **reference** – the discrete input reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_discrete_input_flag(reference: int) → bool
```

Read a bit from discrete input memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_discrete_input_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_discrete_input_nibble(reference: int) → int
```

Read a nibble from discrete input memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_discrete_input_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_extended_memory(memory_number: int, record_number: int, record_length: int) →
List[int]
```

Query and return extended memory from device using Modbus function code 20.

Parameters:

- **memory_number** – the memory number (0x0001 to 0xffff)
- **record_number** – the record to start reading from
- **record_length** – the length of records

Returns:
the read records

```
read_holding_register(reference: int) → int
```

Read a single holding register using Modbus function code 3.

Parameters:
reference – the register reference.
Returns:
the holding register value

```
read_holding_register_bdouble(reference: int) → float
```

Read a double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_bdouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_bfloat(reference: int) → float
```

Read a float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_bfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_bitfield(reference: int, length: int) → int
```

Read a bitfield from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_holding_register_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → ByteStruct | ByteType
```

Read data from holding registers using Modbus function code 3.

Parameters:

- **reference** – the register reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_holding_register_double21436587(reference: int) → float
```

Read a 21436587-endian double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_double21436587_list(reference: int, quantity: int) → List[float]
```

Read a list of 21436587-endian doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_double78563412(reference: int) → float
```

Read a 78563412-endian double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_double78563412_list(reference: int, quantity: int) → List[float]
```

Read a list of 78563412-endian doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_flag(reference: int) → bool
```

Read a bit from holding register memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_holding_register_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_holding_register_float2143(reference: int) → float
```

Read a 2143-endian float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_float2143_list(reference: int, quantity: int) → List[float]
```

Read a list of 2143-endian floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_float3412(reference: int) → float
```

Read a 3412-endian float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_float3412_list(reference: int, quantity: int) → List[float]
```

Read a list of 3412-endian floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_ldouble(reference: int) → float
```

Read a double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_ldouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_lfloat(reference: int) → float
```

Read a float from holding register memory.

Parameters:
reference – the reference to read from
Returns:

float value read from `reference`

```
read_holding_register_lfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_nibble(reference: int) → int
```

Read a nibble from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint16(reference: int) → int
```

Read a signed 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint32(reference: int) → int
```

Read a signed 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint64(reference: int) → int
```

Read a signed 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint8(reference: int) → int
```

Read a signed 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint2143(reference: int) → int
```

Read a signed 2143-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint21436587(reference: int) → int
```

Read a signed 21436587-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 21436587-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:

list of int values read from `reference`

```
read_holding_register_sint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 2143-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint3412(reference: int) → int
```

Read a signed 3412-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 3412-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint78563412(reference: int) → int
```

Read a signed 78563412-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 78563412-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint16(reference: int) → int
```

Read a signed 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint32(reference: int) → int
```

Read a signed 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint64(reference: int) → int
```

Read a signed 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint8(reference: int) → int
```

Read a signed 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_string(reference: int, length: int) → bytes
```

Read a string from holding register memory.

Parameters:

- **reference** – the reference to read from

- **length** – the length of the value

Returns:
bytes value read from `reference`

```
read_holding_register_string_list(reference: int, quantity: int, length: int) →
List[bytes]
```

Read a list of strings from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of bytes values read from `reference`

```
read_holding_register_ubint16(reference: int) → int
```

Read an unsigned 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint32(reference: int) → int
```

Read an unsigned 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint64(reference: int) → int
```

Read an unsigned 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint8(reference: int) → int
```

Read an unsigned 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint2143(reference: int) → int
```

Read an unsigned 2143-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint21436587(reference: int) → int
```

Read an unsigned 21436587-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 21436587-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 2143-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint3412(reference: int) → int
```

Read an unsigned 3412-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 3412-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint78563412(reference: int) → int
```

Read an unsigned 78563412-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 78563412-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint16(reference: int) → int
```

Read an unsigned 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint32(reference: int) → int
```

Read an unsigned 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint64(reference: int) → int
```

Read an unsigned 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint8(reference: int) → int
```

Read an unsigned 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register(reference: int) → int
```

Read a single input register using Modbus function code 4.

Parameters:
reference – the register reference.
Returns:
the input register value

```
read_input_register_bdouble(reference: int) → float
```

Read a double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_bdouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_bfloat(reference: int) → float
```

Read a float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_bfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_bitfield(reference: int, length: int) → int
```

Read a bitfield from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_input_register_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct]
| Type[ByteType]) → ByteStruct | ByteType
```

Read data from input registers using Modbus function code 4.

Parameters:

- **reference** – the register reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_input_register_double21436587(reference: int) → float
```

Read a 21436587-endian double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_double21436587_list(reference: int, quantity: int) → List[float]
```

Read a list of 21436587-endian doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_double78563412(reference: int) → float
```

Read a 78563412-endian double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_double78563412_list(reference: int, quantity: int) → List[float]
```

Read a list of 78563412-endian doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_flag(reference: int) → bool
```

Read a bit from input register memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_input_register_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_input_register_float2143(reference: int) → float
```

Read a 2143-endian float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_float2143_list(reference: int, quantity: int) → List[float]
```

Read a list of 2143-endian floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_float3412(reference: int) → float
```

Read a 3412-endian float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_float3412_list(reference: int, quantity: int) → List[float]
```

Read a list of 3412-endian floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_ldouble(reference: int) → float
```

Read a double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_ldouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_lfloat(reference: int) → float
```

Read a float from input register memory.

Parameters:
reference – the reference to read from
Returns:

float value read from `reference`

```
read_input_register_lfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_nibble(reference: int) → int
```

Read a nibble from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint16(reference: int) → int
```

Read a signed 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint32(reference: int) → int
```

Read a signed 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint64(reference: int) → int
```

Read a signed 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint8(reference: int) → int
```

Read a signed 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint2143(reference: int) → int
```

Read a signed 2143-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint21436587(reference: int) → int
```

Read a signed 21436587-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 21436587-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:

list of int values read from `reference`

```
read_input_register_sint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 2143-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint3412(reference: int) → int
```

Read a signed 3412-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 3412-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint78563412(reference: int) → int
```

Read a signed 78563412-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 78563412-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint16(reference: int) → int
```

Read a signed 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint32(reference: int) → int
```

Read a signed 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint64(reference: int) → int
```

Read a signed 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint8(reference: int) → int
```

Read a signed 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_string(reference: int, length: int) → bytes
```

Read a string from input register memory.

Parameters:

- **reference** – the reference to read from

- **length** – the length of the value

Returns:
bytes value read from `reference`

```
read_input_register_string_list(reference: int, quantity: int, length: int) →
List[bytes]
```

Read a list of strings from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of bytes values read from `reference`

```
read_input_register_ubint16(reference: int) → int
```

Read an unsigned 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint32(reference: int) → int
```

Read an unsigned 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint64(reference: int) → int
```

Read an unsigned 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint8(reference: int) → int
```

Read an unsigned 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint2143(reference: int) → int
```

Read an unsigned 2143-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint21436587(reference: int) → int
```

Read an unsigned 21436587-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 21436587-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 2143-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint3412(reference: int) → int
```

Read an unsigned 3412-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 3412-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint78563412(reference: int) → int
```

Read an unsigned 78563412-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 78563412-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint16(reference: int) → int
```

Read an unsigned 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint32(reference: int) → int
```

Read an unsigned 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint64(reference: int) → int
```

Read an unsigned 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint8(reference: int) → int
```

Read an unsigned 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_multiple_coils(reference: int, quantity: int) → List[int]
```

Read multiple coils using Modbus function code 1.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of coils to be read.

Returns:
a list of coil states where each element is `1` if the corresponding coil is set and `0` else.

```
read_multiple_discrete_inputs(reference: int, quantity: int) → List[int]
```

Read multiple discrete inputs using Modbus function code 2.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of discrete inputs to be read.

Returns:
a list of discrete inputs where each element is `1` if the corresponding discrete input is set and `0` else.

```
read_multiple_holding_registers(reference: int, quantity: int) → List[int]
```

Read multiple holding registers using Modbus function code 3.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of registers to be read.

Returns:
the holding register values

```
read_multiple_input_registers(reference: int, quantity: int) → List[int]
```

Read multiple input registers using Modbus function code 4.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of registers to be read.

Returns:
the input register values

```
read_write_multiple_holding_registers(read_reference: int, read_quantity: int,
write_reference: int, write_values: List[int]) → List[int]
```

Read/Write multiple holding registers using Modbus function code 23.

Parameters:

- **read_reference** – the register reference to read from.
- **read_quantity** – the number of registers to read.
- **write_reference** – the register reference to to write to.
- **write_values** – the values to be written.

Returns:
the read registers

```
reconnect() → None
```

Reconnect socket.

```
write(adu: SerialRTUModbusRequest | SerialASCIIModbusRequest | TCPModbusRequest,
auto_length: bool = True) → None
```

Write `adu` encoded using the interface. The transaction identifier is set automatically. The unit identifier is put into the adu automatically, too.

Parameters:

- **adu** – a serializable adu
- **auto_length=True** – if set to `True` the length is calculated automatically

```
write_coil(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a single coil using Modbus function code 5.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written.

```
write_coil_bitfield(reference: int, value: int, length: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_coil_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_coil_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct] |
Type[ByteType]) → SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write data to coils using Modbus function code 15.

Parameters:

- **reference** – the coil reference.
- **data** – the type or instance or the decoder for the data

```
write_coil_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_coil_flag_list(reference: int, values: List[bool]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_coil_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_coil_nibble_list(reference: int, values: List[int]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_extended_memory(memory_number: int, record_number: int, records: List[int]) → None
```

Write extended memory using Modbus function code 21.

Parameters:

- **memory_number** – the memory number (0x0001 to 0xffff)
- **record_number** – the record to write to
- **records** – the records to be written

```
write_holding_register(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write single holding register using Modbus function code 6.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written into the corresponding register.

```
write_holding_register_bdouble(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_bdouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_bfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_bfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_bitfield(reference: int, value: int, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_holding_register_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_holding_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → SerialRTUModbusResponse | SerialASCIIModbusResponse |
TCPModbusResponse
```

Write data to holding registers using Modbus function code 16.

Parameters:

- **reference** – the register reference.
- **data** – the type or instance or the decoder for the data

```
write_holding_register_double21436587(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 21436587-endian double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_double21436587_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 21436587-endian doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_double78563412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 78563412-endian double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_double78563412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 78563412-endian doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_holding_register_flag_list(reference: int, values: List[bool]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_holding_register_float2143(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 2143-endian float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_float2143_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 2143-endian floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_float3412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 3412-endian float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_float3412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 3412-endian floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_ldouble(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_ldouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_lfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_lfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_nibble_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 2143-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 21436587-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 21436587-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 2143-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 3412-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 3412-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 78563412-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 78563412-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_string(reference: int, value: bytes, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a string to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bytes value to be written
- **length** – the length of the value

```
write_holding_register_string_list(reference: int, values: List[bytes], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of strings to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bytes values to be written
- **length** – the length of the value

```
write_holding_register_ubint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 2143-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 21436587-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 21436587-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 2143-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 3412-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 3412-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 78563412-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 78563412-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_multiple_coils(reference: int, values: List[int]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write multiple coils using Modbus function code 15.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

```
write_multiple_holding_registers(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write multiple holding registers using Modbus function code 16.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

UDP Client

```
class htf.modbus.ModbusUDPClient(address: str, port: int = 502, unit_identifier: int = 0,
timeout: int | float = 1.0, retries: int | None = None, references_start_at_one: bool = True,
transaction_identifier: int = 0, request_parser: Type[SerialRTUModbusRequest] |
Type[SerialASCIIModbusRequest] | Type[TCPModbusRequest] = <class
'htf.modbus.protocol.TCPModbusRequest'>, response_parser: Type[SerialRTUModbusResponse] |
Type[SerialASCIIModbusResponse] | Type[TCPModbusResponse] = <class
'htf.modbus.protocol.TCPModbusResponse'>, debuglevel: int | None = None)
```

Parameters:

- **address** – the ip address to connect to
- **port** – the port to be used
- **unit_identifier** – the unit identifier for Modbus TCP
- **timeout** – the timeout in seconds
- **retries=None** – the number of retries until an exception is raised
- **references_start_at_one=True** – if set to `True` all references start at `1` else `0`
- **transaction_identifier** – the initial transaction identifier
- **request_parser** – a parser for requests
- **response_parser** – a parser for responses
- **debuglevel** – the debuglevel (`None` disables debug, `1` prints requests and responses and `2` prints introspection)

```
build_request() → SerialRTUModbusRequest | SerialASCIIModbusRequest | TCPModbusRequest
```

Build and return an instance of the desired request parser.

Returns:
request parser

```
build_response() → SerialRTUModbusResponse | SerialASCIIModbusResponse |
TCPModbusResponse
```

Build and return an instance of the desired response parser.

Returns:
response parser

```
close() → None
```

Close the interface.

```
get_device_identification() → Dict[int, bytes]
```

Query and return the device identification using Modbus function code 43.14.

Returns:
containing key values pairs with the device identifiers

```
get_fifo_queue_content(address: int) → List[int]
```

Query and return the fifo content of `address` using Modbus function code 24.

Parameters:
address – the fifo queue pointer address
Returns:
list of content from the fifo queue at `address`

```
get_transaction_identifier() → int
```

Create and return a new transaction identifer. Is thread-safe.

Returns:
the next transaction identifier

```
mask_holding_register(reference: int, and_mask: int, or_mask: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Mask write single holding register using Modbus function code 22.

Parameters:

- **reference** – the register reference

- **and_mask** – the and mask
- **or_mask** – the or mask

```
mask_multiple_holding_registers(reference: int, quantity: int, and_mask: int, or_mask: int) → None
```

Mask write multiple holding registers using Modbus function code 16.

Parameters:

- **reference** – the register reference
- **quantity** – the number of registers to be masked
- **and_mask** – the and mask
- **or_mask** – the or mask

```
query(adu: SerialRTUModbusRequest | SerialASCIIModbusRequest | TCPModbusRequest) → SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Query `adu` using the interface and return the decoded response.

Parameters:
adu – a serializable adu
Returns:
the decoded response if no exception occurred
Return type:
adu
Raises:
ModbusException – in case of an exception response

```
read() → SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Read a message from the interface and decode it.

Returns:
the decoded adu as is
Return type:
adu
Raises:
Exception if no answer was received –

```
read_coil(reference: int) → int
```

Read a single coil using Modbus function code 1.

Parameters:
reference – the register reference.
Returns:

`1` if the coil is set and `0` else.

```
read_coil_bitfield(reference: int, length: int) → int
```

Read a bitfield from coil memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_coil_bitfield_list(reference: int, quantity: int, length: int) → List[int]
```

Read a list of bitfields from coil memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_coil_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct] |
Type[ByteType]) → ByteStruct | ByteType
```

Read data from coils using Modbus function code 1.

Parameters:

- **reference** – the coil reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_coil_flag(reference: int) → bool
```

Read a bit from coil memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_coil_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from coil memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_coil_nibble(reference: int) → int
```

Read a nibble from coil memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_coil_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from coil memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_discrete_input(reference: int) → int
```

Read single discrete input using Modbus function code 2.

Parameters:
reference – the register reference.
Returns:
`1` if the discrete input is set and `0` else.

```
read_discrete_input_bitfield(reference: int, length: int) → int
```

Read a bitfield from discrete input memory.

Parameters:

- **reference** – the reference to read from

- **length** – the length of the value

Returns:
int value read from `reference`

```
read_discrete_input_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_discrete_input_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct]
| Type[ByteType]) → ByteStruct | ByteType
```

Read data from discrete inputs using Modbus function code 2.

Parameters:

- **reference** – the discrete input reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_discrete_input_flag(reference: int) → bool
```

Read a bit from discrete input memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_discrete_input_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_discrete_input_nibble(reference: int) → int
```

Read a nibble from discrete input memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_discrete_input_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_extended_memory(memory_number: int, record_number: int, record_length: int) →
List[int]
```

Query and return extended memory from device using Modbus function code 20.

Parameters:

- **memory_number** – the memory number (0x0001 to 0xffff)
- **record_number** – the record to start reading from
- **record_length** – the length of records

Returns:
the read records

```
read_holding_register(reference: int) → int
```

Read a single holding register using Modbus function code 3.

Parameters:
reference – the register reference.
Returns:
the holding register value

```
read_holding_register_bdouble(reference: int) → float
```

Read a double from holding register memory.

Parameters:

reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_bdouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_bfloat(reference: int) → float
```

Read a float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_bfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_bitfield(reference: int, length: int) → int
```

Read a bitfield from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_holding_register_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → ByteStruct | ByteType
```

Read data from holding registers using Modbus function code 3.

Parameters:

- **reference** – the register reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_holding_register_double21436587(reference: int) → float
```

Read a 21436587-endian double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_double21436587_list(reference: int, quantity: int) → List[float]
```

Read a list of 21436587-endian doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_double78563412(reference: int) → float
```

Read a 78563412-endian double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_double78563412_list(reference: int, quantity: int) → List[float]
```

Read a list of 78563412-endian doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_flag(reference: int) → bool
```

Read a bit from holding register memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_holding_register_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_holding_register_float2143(reference: int) → float
```

Read a 2143-endian float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_float2143_list(reference: int, quantity: int) → List[float]
```

Read a list of 2143-endian floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_float3412(reference: int) → float
```

Read a 3412-endian float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_float3412_list(reference: int, quantity: int) → List[float]
```

Read a list of 3412-endian floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_ldouble(reference: int) → float
```

Read a double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_ldouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from holding register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_lfloat(reference: int) → float
```

Read a float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_lfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_nibble(reference: int) → int
```

Read a nibble from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint16(reference: int) → int
```

Read a signed 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint32(reference: int) → int
```

Read a signed 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint64(reference: int) → int
```

Read a signed 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint8(reference: int) → int
```

Read a signed 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint2143(reference: int) → int
```

Read a signed 2143-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint21436587(reference: int) → int
```

Read a signed 21436587-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:

int value read from `reference`

```
read_holding_register_sint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 21436587-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 2143-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint3412(reference: int) → int
```

Read a signed 3412-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 3412-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint78563412(reference: int) → int
```

Read a signed 78563412-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 78563412-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint16(reference: int) → int
```

Read a signed 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint32(reference: int) → int
```

Read a signed 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint64(reference: int) → int
```

Read a signed 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint8(reference: int) → int
```

Read a signed 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_string(reference: int, length: int) → bytes
```

Read a string from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
bytes value read from `reference`

```
read_holding_register_string_list(reference: int, quantity: int, length: int) →
List[bytes]
```

Read a list of strings from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of bytes values read from `reference`

```
read_holding_register_ubint16(reference: int) → int
```

Read an unsigned 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint32(reference: int) → int
```

Read an unsigned 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint64(reference: int) → int
```

Read an unsigned 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint8(reference: int) → int
```

Read an unsigned 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint2143(reference: int) → int
```

Read an unsigned 2143-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint21436587(reference: int) → int
```

Read an unsigned 21436587-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 21436587-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 2143-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint3412(reference: int) → int
```

Read an unsigned 3412-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 3412-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint78563412(reference: int) → int
```

Read an unsigned 78563412-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 78563412-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint16(reference: int) → int
```

Read an unsigned 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:

int value read from `reference`

```
read_holding_register_ulint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint32(reference: int) → int
```

Read an unsigned 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint64(reference: int) → int
```

Read an unsigned 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint8(reference: int) → int
```

Read an unsigned 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register(reference: int) → int
```

Read a single input register using Modbus function code 4.

Parameters:
reference – the register reference.
Returns:
the input register value

```
read_input_register_bdouble(reference: int) → float
```

Read a double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_bdouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_bfloat(reference: int) → float
```

Read a float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_bfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_bitfield(reference: int, length: int) → int
```

Read a bitfield from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_input_register_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct]
| Type[ByteType]) → ByteStruct | ByteType
```

Read data from input registers using Modbus function code 4.

Parameters:

- **reference** – the register reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_input_register_double21436587(reference: int) → float
```

Read a 21436587-endian double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_double21436587_list(reference: int, quantity: int) → List[float]
```

Read a list of 21436587-endian doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_double78563412(reference: int) → float
```

Read a 78563412-endian double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_double78563412_list(reference: int, quantity: int) → List[float]
```

Read a list of 78563412-endian doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_flag(reference: int) → bool
```

Read a bit from input register memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_input_register_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_input_register_float2143(reference: int) → float
```

Read a 2143-endian float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_float2143_list(reference: int, quantity: int) → List[float]
```

Read a list of 2143-endian floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_float3412(reference: int) → float
```

Read a 3412-endian float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_float3412_list(reference: int, quantity: int) → List[float]
```

Read a list of 3412-endian floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_ldouble(reference: int) → float
```

Read a double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_ldouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_lfloat(reference: int) → float
```

Read a float from input register memory.

Parameters:
reference – the reference to read from
Returns:

float value read from `reference`

```
read_input_register_lfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_nibble(reference: int) → int
```

Read a nibble from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint16(reference: int) → int
```

Read a signed 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint32(reference: int) → int
```

Read a signed 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint64(reference: int) → int
```

Read a signed 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint8(reference: int) → int
```

Read a signed 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint2143(reference: int) → int
```

Read a signed 2143-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint21436587(reference: int) → int
```

Read a signed 21436587-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 21436587-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:

list of int values read from `reference`

```
read_input_register_sint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 2143-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint3412(reference: int) → int
```

Read a signed 3412-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 3412-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint78563412(reference: int) → int
```

Read a signed 78563412-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 78563412-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint16(reference: int) → int
```

Read a signed 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint32(reference: int) → int
```

Read a signed 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint64(reference: int) → int
```

Read a signed 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint8(reference: int) → int
```

Read a signed 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_string(reference: int, length: int) → bytes
```

Read a string from input register memory.

Parameters:

- **reference** – the reference to read from

- **length** – the length of the value

Returns:
bytes value read from `reference`

```
read_input_register_string_list(reference: int, quantity: int, length: int) →
List[bytes]
```

Read a list of strings from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of bytes values read from `reference`

```
read_input_register_ubint16(reference: int) → int
```

Read an unsigned 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint32(reference: int) → int
```

Read an unsigned 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint64(reference: int) → int
```

Read an unsigned 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint8(reference: int) → int
```

Read an unsigned 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint2143(reference: int) → int
```

Read an unsigned 2143-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint21436587(reference: int) → int
```

Read an unsigned 21436587-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 21436587-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 2143-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint3412(reference: int) → int
```

Read an unsigned 3412-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 3412-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint78563412(reference: int) → int
```

Read an unsigned 78563412-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 78563412-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint16(reference: int) → int
```

Read an unsigned 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint32(reference: int) → int
```

Read an unsigned 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint64(reference: int) → int
```

Read an unsigned 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint8(reference: int) → int
```

Read an unsigned 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_multiple_coils(reference: int, quantity: int) → List[int]
```

Read multiple coils using Modbus function code 1.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of coils to be read.

Returns:
a list of coil states where each element is `1` if the corresponding coil is set and `0` else.

```
read_multiple_discrete_inputs(reference: int, quantity: int) → List[int]
```

Read multiple discrete inputs using Modbus function code 2.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of discrete inputs to be read.

Returns:
a list of discrete inputs where each element is `1` if the corresponding discrete input is set and `0` else.

```
read_multiple_holding_registers(reference: int, quantity: int) → List[int]
```

Read multiple holding registers using Modbus function code 3.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of registers to be read.

Returns:
the holding register values

```
read_multiple_input_registers(reference: int, quantity: int) → List[int]
```

Read multiple input registers using Modbus function code 4.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of registers to be read.

Returns:
the input register values

```
read_write_multiple_holding_registers(read_reference: int, read_quantity: int,
write_reference: int, write_values: List[int]) → List[int]
```

Read/Write multiple holding registers using Modbus function code 23.

Parameters:

- **read_reference** – the register reference to read from.
- **read_quantity** – the number of registers to read.
- **write_reference** – the register reference to to write to.
- **write_values** – the values to be written.

Returns:
the read registers

```
reconnect() → None
```

Reconnect socket.

```
write(adu: SerialRTUModbusRequest | SerialASCIIModbusRequest | TCPModbusRequest,
auto_length: bool = True) → None
```

Write `adu` encoded using the interface. The transaction identifier is set automatically. The unit identifier is put into the adu automatically, too.

Parameters:

- **adu** – a serializable adu
- **auto_length=True** – if set to `True` the length is calculated automatically

```
write_coil(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a single coil using Modbus function code 5.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written.

```
write_coil_bitfield(reference: int, value: int, length: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_coil_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_coil_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct] |
Type[ByteType]) → SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write data to coils using Modbus function code 15.

Parameters:

- **reference** – the coil reference.
- **data** – the type or instance or the decoder for the data

```
write_coil_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_coil_flag_list(reference: int, values: List[bool]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_coil_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_coil_nibble_list(reference: int, values: List[int]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_extended_memory(memory_number: int, record_number: int, records: List[int]) → None
```

Write extended memory using Modbus function code 21.

Parameters:

- **memory_number** – the memory number (0x0001 to 0xffff)
- **record_number** – the record to write to
- **records** – the records to be written

```
write_holding_register(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write single holding register using Modbus function code 6.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written into the corresponding register.

```
write_holding_register_bdouble(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_bdouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_bfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_bfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_bitfield(reference: int, value: int, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_holding_register_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_holding_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → SerialRTUModbusResponse | SerialASCIIModbusResponse |
TCPModbusResponse
```

Write data to holding registers using Modbus function code 16.

Parameters:

- **reference** – the register reference.
- **data** – the type or instance or the decoder for the data

```
write_holding_register_double21436587(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 21436587-endian double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_double21436587_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 21436587-endian doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_double78563412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 78563412-endian double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_double78563412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 78563412-endian doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_holding_register_flag_list(reference: int, values: List[bool]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_holding_register_float2143(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 2143-endian float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_float2143_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 2143-endian floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_float3412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 3412-endian float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_float3412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 3412-endian floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_ldouble(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_ldouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_lfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_lfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_nibble_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 2143-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 21436587-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 21436587-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 2143-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 3412-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 3412-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 78563412-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 78563412-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_string(reference: int, value: bytes, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a string to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bytes value to be written
- **length** – the length of the value

```
write_holding_register_string_list(reference: int, values: List[bytes], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of strings to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bytes values to be written
- **length** – the length of the value

```
write_holding_register_ubint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 2143-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 21436587-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 21436587-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 2143-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 3412-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 3412-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 78563412-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 78563412-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_multiple_coils(reference: int, values: List[int]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write multiple coils using Modbus function code 15.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

```
write_multiple_holding_registers(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write multiple holding registers using Modbus function code 16.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

Servers

Servers are used to respond to queries initiated by clients. Servers are used by devices. Users can add own servers using own interfaces.

Base Server

```
class htf.modbus.ModbusBaseServer
```

Modbus Base Server is an abstract base class.

```
clear_counters() → None
```

Clear all counters.

```
clear_diagnostics() → None
```

Clear diagnostics.

```
close() → None
```

Close interface and stop the server.

```
handle_request(data: bytes) → None
```

Request handler.

Raises:
Exception – abstract base class has no implementation.

```
set_listen_only_mode(listen_only_mode_enabled: bool = True) → None
```

Set listen only mode.

Parameters:
listen_only_mode_enabled – `True` enables the listen only mode. `False` disables it.

```
start() → None
```

Start the server.

Raises:
Exception – abstract base class has no implementation.

```
stop() → None
```

Stop the server.

Raises:
Exception – abstract base class has no implementation.

Serial RTU Server

```
class htf.modbus.ModbusSerialRTUServer(com_port: str, baud: int = 9600, bytesize: int = 8,
parity: str = 'E', stopbits: int = 1, rtscts: bool = False, device_address: int = 1, timeout:
int | float = 1.0, request_parser: ByteStruct | BitStruct | ByteType | BitType = <class
'htf.modbus.protocol.SerialRTUModbusRequest'>, response_parser: ByteStruct | BitStruct |
ByteType | BitType = <class 'htf.modbus.protocol.SerialRTUModbusResponse'>, callback:
Callable | None = None)
```

Parameters:

- **com_port** – the com port to be used
- **baud** – the baud rate
- **bytesize** – number of bits in a byte
- **parity** – serial.PARITY_NONE, serial.PARITY_ODD or serial.PARITY_EVEN
- **stopbits** – number of stopbits (1, 1.5 or 2)
- **rtscts** – if True use flow control
- **device_address** – the device address to be used (Slave ID)

- **timeout** – the timeout in seconds
- **request_parser** – the request parser
- **response_parser** – the response parser
- **callback** – a callback used as a request handler

```
clear_counters() → None
```

Clear all counters.

```
clear_diagnostics() → None
```

Clear diagnostics.

```
close() → None
```

Close interface and stop the server.

```
disable_broken_checksums() → None
```

Disable broken checksums for testing purposes.

```
enable_broken_checksums() → None
```

Enable broken checksums for testing purposes.

```
static get_broadcast_enabled_function_codes() → List[int]
```

Return a list of functions codes that are processed for broadcast requests. By default the function codes `5`, `6`, `15` and `16` are supported. This method can be overwritten easily.

Returns:
a list of int containing the supported function codes

```
handle_request(data: bytes) → None
```

Request handler. A request is handled by calling the callback.

Parameters:
data – a request to be handled

```
set_listen_only_mode(listen_only_mode_enabled: bool = True) → None
```

Set listen only mode.

Parameters:
listen_only_mode_enabled – `True` enables the listen only mode. `False` disables it.

```
set_receive_event_callback(callback: Callable | None) → None
```

Set the callback to emit events.

Parameters:
callback – the callback to be called

```
start() → None
```

Start the server.

```
stop() → None
```

Stop the server.

Serial ASCII Server

```
class htf.modbus.ModbusSerialASCIIServer(com_port: str, baud: int = 9600,
bytesize: int = 8, parity: str = 'E', stopbits: int = 1, rtscts: bool = False,
device_address: int = 1, timeout: int | float = 1.0, request_parser: ByteStruct | BitStruct |
ByteType | BitType = <class 'htf.modbus.protocol.SerialASCIIModbusRequest'>, response_parser:
ByteStruct | BitStruct | ByteType | BitType = <class
'htf.modbus.protocol.SerialASCIIModbusResponse'>, callback: Callable | None = None)
```

Parameters:

- **com_port** – the com port to be used
- **baud** – the baud rate
- **bytesize** – number of bits in a byte
- **parity** – serial.PARITY_NONE, serial.PARITY_ODD or serial.PARITY_EVEN
- **stopbits** – number of stopbits (1, 1.5 or 2)
- **rtscts** – if True use flow control
- **device_address** – the device address to be used (Slave ID)
- **timeout** – the timeout in seconds
- **request_parser** – the request parser
- **response_parser** – the response parser
- **callback** – a callback used as a request handler

```
change_ascii_delimiter(delimiter: bytes) → None
```

Change ASCII delimiter to `ascii_delimiter`.

Parameters:
delimiter – bytes of length 1 representing the new lf character

```
clear_counters() → None
```

Clear all counters.

```
clear_diagnostics() → None
```

Clear diagnostics.

```
close() → None
```

Close interface and stop the server.

```
disable_broken_checksums() → None
```

Disable broken checksums for testing purposes.

```
enable_broken_checksums() → None
```

Enable broken checksums for testing purposes.

```
static get_broadcast_enabled_function_codes() → List[int]
```

Return a list of functions codes that are processed for broadcast requests. By default the function codes `5`, `6`, `15` and `16` are supported. This method can be overwritten easily.

Returns:
a list of int containing the supported function codes

```
handle_request(data: bytes) → None
```

Request handler. A request is handled by calling the callback.

Parameters:
data – a request to be handled

```
set_listen_only_mode(listen_only_mode_enabled: bool = True) → None
```

Set listen only mode.

Parameters:
listen_only_mode_enabled – `True` enables the listen only mode. `False` disables it.

```
set_receive_event_callback(callback: Callable | None) → None
```

Set the callback to emit events.

Parameters:
callback – the callback to be called

```
start() → None
```

Start the server.

```
stop() → None
```

Stop the server.

Device Simulators

Devices are used to simulate Modbus devices. Users can add own devices with own commands easily.

Base Device

```
class htf.modbus.ModbusBaseDevice(server: ModbusBaseServer, memory: ModbusMemory,
debuglevel: int | None = None)
```

Parameters:

- **server** – a server instance to be used
- **memory** – a memory instance to be used
- **debuglevel** – the debuglevel (`None` disables debug, `1` prints requests and responses and `2` prints introspection)

```
close() → None
```

Close the device simulator and the server.

```
create_exception_response(request_pdu: RequestPDU, exception_code: str) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler to response with an exception.

Parameters:

- **request_pdu** – the request pdu
- **exception_code** – the exception code to be sent

Returns:
the response to be sent to the client

```
get_extended_memory(memory_number: int) → Dict[int, List[int]] | None
```

Get extended memory.

Parameters:
memory_number – the memory number (0x0001 to 0xFFFF)
Returns:
the extended memory at `memory_number` as a dict with key record numer and values list of
integers

Return type:
list of int
Raises:
KeyError – if the `memory_number` was not found in extended memory.

```
handle_function_code_1(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 1 ``Read Coils`.``.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_15(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 15 `Write Multiple Coils`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_16(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 16 `Write Multiple Registers`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_2(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 2 `Read Discrete Inputs`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_20(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 20 `Read File Record`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_21(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 21 `Write File Record`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_22(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 22 `Mask Write Register`.

Parameters:
pdu – the request pdu

Returns:
the response to be sent to the client

```
handle_function_code_23(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 23 `Read/Write Multiple Registers`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_24(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 24 `Read FIFO Queue`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_3(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 3 `Read Holding Registers`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_4(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 4 `Read Input Registers`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_43(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Encapsulated Interface Transport.

Calls a handler by mei type with the name `handle_function_code_43_mei_type_<mei type>`. To add more mei types more handlers have to be added.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_43_mei_type_14(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Encapsulated Interface Transport: Read Device Identification. Basic object ids can be read.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_5(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 5 `Write Single Coil`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_6(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 6 `Write Single Register`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_request(adu: SerialRTUModbusRequest | SerialASCIIModbusRequest | TCPModbusRequest)
→ SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for an incoming request.

Parameters:
adu – a request adu
Returns:

the command or exception handler

```
mask_holding_register(reference: int, and_mask: int, or_mask: int) → None
```

Mask write single holding register from device memory.

Parameters:

- **reference** – the register reference
- **and_mask** – the and mask
- **or_mask** – the or mask

```
mask_multiple_holding_registers(reference: int, quantity: int, and_mask: int, or_mask:
int) → None
```

Mask write multiple holding registers from device memory.

Parameters:

- **reference** – the register reference
- **quantity** – the number of registers to be masked
- **and_mask** – the and mask
- **or_mask** – the or mask

```
read_coil(reference: int) → int
```

Read a single coil from device memory.

Parameters:
reference – the register reference.
Returns:
`1` if the coil is set and `0` else.

```
read_coil_bitfield(reference: int, length: int) → int
```

Read a bitfield from coil memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_coil_bitfield_list(reference: int, quantity: int, length: int) → List[int]
```

Read a list of bitfields from coil memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_coil_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct] |
Type[ByteType]) → ByteStruct | ByteType
```

Read data from coils device memory.

Parameters:

- **reference** – the coil reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_coil_flag(reference: int) → bool
```

Read a bit from coil memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_coil_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from coil memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_coil_nibble(reference: int) → int
```

Read a nibble from coil memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_coil_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from coil memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_discrete_input(reference: int) → int
```

Read single discrete input from device memory

Parameters:
reference – the register reference.
Returns:
`1` if the discrete input is set and `0` else.

```
read_discrete_input_bitfield(reference: int, length: int) → int
```

Read a bitfield from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_discrete_input_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_discrete_input_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct]
| Type[ByteType]) → ByteStruct | ByteType
```

Read data from discrete inputs device memory.

Parameters:

- **reference** – the discrete input reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_discrete_input_flag(reference: int) → bool
```

Read a bit from discrete input memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_discrete_input_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_discrete_input_nibble(reference: int) → int
```

Read a nibble from discrete input memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_discrete_input_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register(reference: int) → int
```

Read a single holding register from device memory.

Parameters:
reference – the register reference.
Returns:
the holding register value

```
read_holding_register_bdouble(reference: int) → float
```

Read a double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_bdouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_bfloat(reference: int) → float
```

Read a float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_bfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:

list of float values read from `reference`

```
read_holding_register_bitfield(reference: int, length: int) → int
```

Read a bitfield from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_holding_register_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → ByteStruct | ByteType
```

Read data from holding registers device memory.

Parameters:

- **reference** – the register reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_holding_register_double21436587(reference: int) → float
```

Read a 21436587-endian double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_double21436587_list(reference: int, quantity: int) → List[float]
```

Read a list of 21436587-endian doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_double78563412(reference: int) → float
```

Read a 78563412-endian double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_double78563412_list(reference: int, quantity: int) → List[float]
```

Read a list of 78563412-endian doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_flag(reference: int) → bool
```

Read a bit from holding register memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_holding_register_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from holding register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_holding_register_float2143(reference: int) → float
```

Read a 2143-endian float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_float2143_list(reference: int, quantity: int) → List[float]
```

Read a list of 2143-endian floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_float3412(reference: int) → float
```

Read a 3412-endian float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_float3412_list(reference: int, quantity: int) → List[float]
```

Read a list of 3412-endian floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_ldouble(reference: int) → float
```

Read a double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_ldouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_lfloat(reference: int) → float
```

Read a float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_lfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_nibble(reference: int) → int
```

Read a nibble from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint16(reference: int) → int
```

Read a signed 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint32(reference: int) → int
```

Read a signed 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint64(reference: int) → int
```

Read a signed 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint8(reference: int) → int
```

Read a signed 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint2143(reference: int) → int
```

Read a signed 2143-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint21436587(reference: int) → int
```

Read a signed 21436587-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 21436587-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 2143-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint3412(reference: int) → int
```

Read a signed 3412-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 3412-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint78563412(reference: int) → int
```

Read a signed 78563412-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 78563412-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint16(reference: int) → int
```

Read a signed 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint32(reference: int) → int
```

Read a signed 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint64(reference: int) → int
```

Read a signed 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint8(reference: int) → int
```

Read a signed 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_string(reference: int, length: int) → bytes
```

Read a string from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
bytes value read from `reference`

```
read_holding_register_string_list(reference: int, quantity: int, length: int) →
List[bytes]
```

Read a list of strings from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of bytes values read from `reference`

```
read_holding_register_ubint16(reference: int) → int
```

Read an unsigned 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from

Returns:
int value read from `reference`

```
read_holding_register_ubint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint32(reference: int) → int
```

Read an unsigned 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint64(reference: int) → int
```

Read an unsigned 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint8(reference: int) → int
```

Read an unsigned 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint2143(reference: int) → int
```

Read an unsigned 2143-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint21436587(reference: int) → int
```

Read an unsigned 21436587-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 21436587-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 2143-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint3412(reference: int) → int
```

Read an unsigned 3412-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 3412-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint78563412(reference: int) → int
```

Read an unsigned 78563412-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 78563412-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint16(reference: int) → int
```

Read an unsigned 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint32(reference: int) → int
```

Read an unsigned 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint64(reference: int) → int
```

Read an unsigned 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint8(reference: int) → int
```

Read an unsigned 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register(reference: int) → int
```

Read a single input register from device memory.

Parameters:
reference – the register reference.
Returns:
the input register value

```
read_input_register_bdouble(reference: int) → float
```

Read a double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_bdouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_bfloat(reference: int) → float
```

Read a float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_bfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_bitfield(reference: int, length: int) → int
```

Read a bitfield from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_input_register_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct]
| Type[ByteType]) → ByteStruct | ByteType
```

Read data from input registers from device memory.

Parameters:

- **reference** – the register reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_input_register_double21436587(reference: int) → float
```

Read a 21436587-endian double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_double21436587_list(reference: int, quantity: int) → List[float]
```

Read a list of 21436587-endian doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_double78563412(reference: int) → float
```

Read a 78563412-endian double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_double78563412_list(reference: int, quantity: int) → List[float]
```

Read a list of 78563412-endian doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_flag(reference: int) → bool
```

Read a bit from input register memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_input_register_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_input_register_float2143(reference: int) → float
```

Read a 2143-endian float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_float2143_list(reference: int, quantity: int) → List[float]
```

Read a list of 2143-endian floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_float3412(reference: int) → float
```

Read a 3412-endian float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_float3412_list(reference: int, quantity: int) → List[float]
```

Read a list of 3412-endian floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_ldouble(reference: int) → float
```

Read a double from input register memory.

Parameters:
reference – the reference to read from
Returns:

float value read from `reference`

```
read_input_register_ldouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_lfloat(reference: int) → float
```

Read a float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_lfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_nibble(reference: int) → int
```

Read a nibble from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint16(reference: int) → int
```

Read a signed 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint32(reference: int) → int
```

Read a signed 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint64(reference: int) → int
```

Read a signed 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint8(reference: int) → int
```

Read a signed 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint2143(reference: int) → int
```

Read a signed 2143-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:

int value read from `reference`

```
read_input_register_sint21436587(reference: int) → int
```

Read a signed 21436587-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 21436587-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 2143-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint3412(reference: int) → int
```

Read a signed 3412-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 3412-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint78563412(reference: int) → int
```

Read a signed 78563412-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 78563412-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint16(reference: int) → int
```

Read a signed 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint32(reference: int) → int
```

Read a signed 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint64(reference: int) → int
```

Read a signed 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint8(reference: int) → int
```

Read a signed 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:

int value read from `reference`

```
read_input_register_slint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_string(reference: int, length: int) → bytes
```

Read a string from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
bytes value read from `reference`

```
read_input_register_string_list(reference: int, quantity: int, length: int) →
List[bytes]
```

Read a list of strings from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of bytes values read from `reference`

```
read_input_register_ubint16(reference: int) → int
```

Read an unsigned 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint32(reference: int) → int
```

Read an unsigned 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint64(reference: int) → int
```

Read an unsigned 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint8(reference: int) → int
```

Read an unsigned 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint2143(reference: int) → int
```

Read an unsigned 2143-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint21436587(reference: int) → int
```

Read an unsigned 21436587-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 21436587-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 2143-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint3412(reference: int) → int
```

Read an unsigned 3412-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 3412-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint78563412(reference: int) → int
```

Read an unsigned 78563412-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 78563412-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint16(reference: int) → int
```

Read an unsigned 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint32(reference: int) → int
```

Read an unsigned 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint64(reference: int) → int
```

Read an unsigned 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint8(reference: int) → int
```

Read an unsigned 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_multiple_coils(reference: int, quantity: int) → List[int]
```

Read multiple coils from device memory.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of coils to be read.

Returns:
a list of coil states where each element is `1` if the corresponding coil is set and `0` else.

```
read_multiple_discrete_inputs(reference: int, quantity: int) → List[int]
```

Read multiple discrete inputs from device memory.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of discrete inputs to be read.

Returns:
a list of discrete inputs where each element is `1` if the corresponding discrete input is set and `0` else.

```
read_multiple_holding_registers(reference: int, quantity: int) → List[int]
```

Read multiple holding registers from device memory.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of registers to be read.

Returns:
the holding register values

```
read_multiple_input_registers(reference: int, quantity: int) → List[int]
```

Read multiple input registers from device memory.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of registers to be read.

Returns:
the input register values

```
set_device_identifier(object_id: int | None, value: str | bytes | None) → None
```

Set a device identifier. The objects `"vendor_name"`, `"product_code"`, `"major_minor_version"`, `"vendor_url"`, `"product_name"`, `"model_name"` and `"user_application_name"` are predefined.

Set `value` to `None` to remove an entry.

Entries can be read using Modbus function code 43.14.

Parameters:

- **object_id** – the object identifier
- **value** – the value to be used

```
set_extended_memory(memory_number: int, records: Dict[int, List[int]] | None) → None
```

Set extended memory that can be read with Modbus function code 20 and written with Modbus function code 21.

If `records` is `None` the memory is removed.

Parameters:

- **memory_number** – the memory number (0x0001 to 0xFFFF)
- **records** – the records to be stored as a dict with key record numer and values list of integers

```
set_fifo_queue_content(address: int, content: List[int] | None) → None
```

Set the FIFO queue content.

Parameters:

- **address** – the address
- **content** – a list of integers or `None` to delete the queue contents

```
write_coil(reference: int, value: int) → None
```

Write a single coil to device memory.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written.

```
write_coil_bitfield(reference: int, value: int, length: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_coil_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_coil_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct] |
Type[ByteType]) → None
```

Write data to coils device memory.

Parameters:

- **reference** – the coil reference.
- **data** – the type or instance or the decoder for the data

```
write_coil_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_coil_flag_list(reference: int, values: List[bool]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_coil_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to coil memory.

Parameters:

- **reference** – the reference to write to

- **value** – the int value to be written

```
write_coil_nibble_list(reference: int, values: List[int]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_discrete_input(reference: int, value: int) → None
```

Read multiple discrete inputs from device memory

Parameters:

- **reference** – the register reference.
- **value** – the value to be written

Returns:
a list of discrete inputs where each element is `1` if the corresponding discrete input is set and `0` else.

```
write_discrete_input_bitfield(reference: int, value: int, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_discrete_input_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_discrete_input_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → None
```

Write data to discrete input device memory.

Parameters:

- **reference** – the discrete input reference.
- **data** – the type or instance or the decoder for the data

```
write_discrete_input_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_discrete_input_flag_list(reference: int, values: List[bool]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_discrete_input_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_discrete_input_nibble_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register(reference: int, value: int) → None
```

Write single holding register to device memory.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written into the corresponding register.

```
write_holding_register_bdouble(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_bdouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_bfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_bfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_bitfield(reference: int, value: int, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_holding_register_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_holding_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → None
```

Write data to holding registers device memory.

Parameters:

- **reference** – the register reference.
- **data** – the type or instance or the decoder for the data

```
write_holding_register_double21436587(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 21436587-endian double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_double21436587_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 21436587-endian doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_double78563412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 78563412-endian double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_double78563412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 78563412-endian doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_holding_register_flag_list(reference: int, values: List[bool]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_holding_register_float2143(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 2143-endian float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_float2143_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 2143-endian floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_float3412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 3412-endian float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_float3412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 3412-endian floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_ldouble(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_ldouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_lfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_lfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_nibble_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 2143-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 21436587-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 21436587-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 2143-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 3412-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 3412-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 78563412-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 78563412-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_string(reference: int, value: bytes, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a string to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bytes value to be written
- **length** – the length of the value

```
write_holding_register_string_list(reference: int, values: List[bytes], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of strings to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bytes values to be written
- **length** – the length of the value

```
write_holding_register_ubint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 2143-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 21436587-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 21436587-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 2143-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 3412-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 3412-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 78563412-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 78563412-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register(reference: int, value: int) → None
```

Write single input register to device memory.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written into the corresponding register.

```
write_input_register_bdouble(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_bdouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_bfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_bfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_bitfield(reference: int, value: int, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_input_register_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_input_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → None
```

Write data to input registers device memory.

Parameters:

- **reference** – the register reference.
- **data** – the type or instance or the decoder for the data

```
write_input_register_double21436587(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 21436587-endian double to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_double21436587_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 21436587-endian doubles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_double78563412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 78563412-endian double to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_double78563412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 78563412-endian doubles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_input_register_flag_list(reference: int, values: List[bool]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_input_register_float2143(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 2143-endian float to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_float2143_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 2143-endian floats to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_float3412(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 3412-endian float to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_float3412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 3412-endian floats to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_ldouble(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_ldouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_lfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_lfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_nibble_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sbint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sbint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sbint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sbint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sbint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sbint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sbint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sbint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 2143-endian 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 21436587-endian 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 21436587-endian 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 2143-endian 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 3412-endian 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 3412-endian 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 78563412-endian 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 78563412-endian 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_slint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_slint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_slint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_slint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_slint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_slint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_slint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_slint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_string(reference: int, value: bytes, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a string to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bytes value to be written
- **length** – the length of the value

```
write_input_register_string_list(reference: int, values: List[bytes], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of strings to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bytes values to be written
- **length** – the length of the value

```
write_input_register_ubint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ubint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ubint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ubint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ubint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ubint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ubint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ubint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_uint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 2143-endian 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_uint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 21436587-endian 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_uint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 21436587-endian 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_uint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 2143-endian 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_uint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 3412-endian 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_uint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 3412-endian 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_uint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 78563412-endian 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_uint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 78563412-endian 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ulint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ulint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ulint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ulint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ulint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ulint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ulint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ulint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_multiple_coils(reference: int, values: List[int]) → None
```

Write multiple coils to device memory.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

```
write_multiple_discrete_inputs(reference: int, values: List[int]) → None
```

Write multiple discrete inputs to device memory.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

```
write_multiple_holding_registers(reference: int, values: List[int]) → None
```

Write multiple holding registers to device memory.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

```
write_multiple_input_registers(reference: int, values: List[int]) → None
```

Write multiple input registers to device memory.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

Serial RTU Device

```
class htf.modbus.ModbusSerialRTUDevice(com_port: str, baud: int = 9600, bytesize: int = 8,
parity: str = 'E', stopbits: int = 1, rtscts: bool = False, device_address: int = 1,
shared_memory: bool = False, request_parser: Type[SerialRTUModbusRequest] |
Type[SerialASCIIModbusRequest] | Type[TCPModbusRequest] = <class
'htf.modbus.protocol.SerialRTUModbusRequest'>, response_parser: Type[SerialRTUModbusResponse]
| Type[SerialASCIIModbusResponse] | Type[TCPModbusResponse] = <class
'htf.modbus.protocol.SerialRTUModbusResponse'>, debuglevel: int | None = None)
```

Parameters:

- **com_port** – the com port to be used

- **baud** – the baud rate
- **bytesize** – number of bits in a byte
- **parity** – serial.PARITY_NONE, serial.PARITY_ODD or serial.PARITY_EVEN
- **stopbits** – number of stopbits (1, 1.5 or 2)
- **rtscts** – if True use flow control
- **device_address** – the device address to be used (Slave ID)
- **shared_memory** – if set to `True` a single shared memory is used
- **request_parser** – the request parser
- **response_parser** – the response parser
- **debuglevel** – the debuglevel (`None` disables debug, `1` prints requests and responses and `2` prints introspection)

```
add_communication_restart_event() → None
```

Add a communication restart event.

```
add_event(event: int) → None
```

Add an event to event log.

Parameters:
event – the event to be added.

```
add_receive_event(is_communication_error: bool = False, is_character_overrun: bool =
False, is_broadcast: bool = False, is_listen_only_mode: bool = False) → None
```

Add a receive event.

Parameters:

- **is_communication_error** – set to `True` if a communication error occurred
- **is_character_overrun** – set to `True` if a character overrun occurred
- **is_broadcast** – set to `True` if a broadcast message was received
- **is_listen_only_mode** – set to `True` if the server is currently in listen only mode

```
add_send_event(read_exception_sent: bool = False, slave_abort_exeception_sent: bool =
False, slave_busy_exception_sent: bool = False, slave_program_nak_exception_sent: bool =
False, write_timeout_occurred: bool = False) → None
```

Add a send event. The current listen only mode is determined automatically.

Parameters:

- **read_exception_sent** – set to `True` if a read exception sent occurred (exception codes 1-3)
- **slave_abort_exeception_sent** – set to `True` if a slave abort exception sent occurred (execption code 4)

- **slave_busy_exception_sent** – set to `True` if a slave busy exception sent occurred (exception codes 5-6)
- **slave_program_nak_exception_sent** – set to `True` if a slave program NAK exception sent occurred (exception code 7)
- **write_timeout_occurred** – set to `True` if a write timeout occurred

```
add_set_listen_only_mode_event() → None
```

Add a listen only mode event.

```
clear_counters_and_diagnostic_register() → None
```

Clear counters and diagnostic register

```
clear_events() → None
```

Clear event log.

```
clear_overrun_counter_and_flag() → None
```

Clear overrun counter and reset the error flag.

```
close() → None
```

Close the device simulator and the server.

```
create_exception_response(request_pdu: RequestPDU, exception_code: str) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler to response with an exception.

Parameters:

- **request_pdu** – the request pdu
- **exception_code** – the exception code to be sent

Returns:
the response to be sent to the client

```
disable_broken_checksums() → None
```

Disable broken checksums for testing purposes.

```
enable_broken_checksums() → None
```

Enable broken checksums for testing purposes.

```
get_bus_character_overrun_count() → int
```

Get the bus character overrun count.

Returns:
the bus character overrun count

```
get_busy_count() → int
```

Return the current busy count.

Returns:
the busy count

```
get_busy_status() → bool
```

Return the current busy status.

Returns:
`True` if the device is busy else `False`

```
get_diagnostic_register() → bytes
```

Return the diagnostic register

Returns:
the content of the diagnostic register

```
get_exception_status() → int
```

Return the current exception status.

Returns:
the current exception status

```
get_extended_memory(memory_number: int) → Dict[int, List[int]] | None
```

Get extended memory.

Parameters:

memory_number – the memory number (0x0001 to 0xFFFF)
Returns:
the extended memory at `memory_number` as a dict with key record numer and values list of
integers

Return type:
list of int
Raises:
KeyError – if the `memory_number` was not found in extended memory.

```
handle_function_code_1(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 1 ``Read Coils`.``.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_11(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 11 `Get Comm Event Counter (Serial Line only)`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_12(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 12 `Get Comm Event Log (Serial Line only)`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_15(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 15 `Write Multiple Coils`.

Parameters:
pdu – the request pdu
Returns:

the response to be sent to the client

```
handle_function_code_16(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 16 `Write Multiple Registers`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_17(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 17 `Report Slave ID (Serial Line only)`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_2(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 2 `Read Discrete Inputs`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_20(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 20 `Read File Record`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_21(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 21 `Write File Record`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_22(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 22 `Mask Write Register`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_23(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 23 `Read/Write Multiple Registers`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_24(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 24 `Read FIFO Queue`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_3(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 3 `Read Holding Registers`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_4(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 4 `Read Input Registers`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_43(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Encapsulated Interface Transport.

Calls a handler by mei type with the name `handle_function_code_43_mei_type_<mei type>`. To add more mei types more handlers have to be added.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_43_mei_type_14(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Encapsulated Interface Transport: Read Device Identification. Basic object ids can be read.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_5(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 5 `Write Single Coil`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_6(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 6 `Write Single Register`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_7(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 7 `Read Exception Status`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.x Diagnostics (Serial Line only).

Calls a handler by sub function code like `handle_function_code_8_sub_function_<sub function code>`. To add more sub functions more handlers have to be added.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_0(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.0 `Diagnostics Send Echo Response`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_1(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.1 `Diagnostics Restart Communications Option`.

Parameters:
pdu – the request pdu

Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_10(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.10
`Diagnostics Clear counters and diagnostic register`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_11(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.11 `Diagnostics Bus message count`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_12(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.12 `Diagnostics Bus communication error count`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_13(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.13 `Diagnostics Bus exception error count`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_14(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.14 `Diagnostics Return slave message count`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_15(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.15 `Diagnostics Slave No Response Count`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_16(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.16 `Diagnostics Slave NAK Count`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_17(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.17 `Diagnostics Slave Busy Count`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_18(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.18 `Diagnostics Slave Character Overrun Count`.

Parameters:
pdu – the request pdu

Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_2(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.2 `Diagnostics Return diagnostic register`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_20(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.20 `Diagnostics`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_3(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.3 `Diagnostics Change ASCII Input Delimiter`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_4(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.4 `Diagnostics Force Listen Only Mode`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_request(adu: SerialRTUModbusRequest | SerialASCIIModbusRequest | TCPModbusRequest)
→ SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for an incoming request.

Parameters:
adu – a request adu
Returns:
the command or exception handler

```
mask_holding_register(reference: int, and_mask: int, or_mask: int) → None
```

Mask write single holding register from device memory.

Parameters:

- **reference** – the register reference
- **and_mask** – the and mask
- **or_mask** – the or mask

```
mask_multiple_holding_registers(reference: int, quantity: int, and_mask: int, or_mask:
int) → None
```

Mask write multiple holding registers from device memory.

Parameters:

- **reference** – the register reference
- **quantity** – the number of registers to be masked
- **and_mask** – the and mask
- **or_mask** – the or mask

```
read_coil(reference: int) → int
```

Read a single coil from device memory.

Parameters:
reference – the register reference.
Returns:
`1` if the coil is set and `0` else.

```
read_coil_bitfield(reference: int, length: int) → int
```

Read a bitfield from coil memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_coil_bitfield_list(reference: int, quantity: int, length: int) → List[int]
```

Read a list of bitfields from coil memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_coil_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct] |
Type[ByteType]) → ByteStruct | ByteType
```

Read data from coils device memory.

Parameters:

- **reference** – the coil reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_coil_flag(reference: int) → bool
```

Read a bit from coil memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_coil_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from coil memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_coil_nibble(reference: int) → int
```

Read a nibble from coil memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_coil_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from coil memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_discrete_input(reference: int) → int
```

Read single discrete input from device memory

Parameters:
reference – the register reference.
Returns:
`1` if the discrete input is set and `0` else.

```
read_discrete_input_bitfield(reference: int, length: int) → int
```

Read a bitfield from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_discrete_input_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_discrete_input_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct]
| Type[ByteType]) → ByteStruct | ByteType
```

Read data from discrete inputs device memory.

Parameters:

- **reference** – the discrete input reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_discrete_input_flag(reference: int) → bool
```

Read a bit from discrete input memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_discrete_input_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_discrete_input_nibble(reference: int) → int
```

Read a nibble from discrete input memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_discrete_input_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from discrete input memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register(reference: int) → int
```

Read a single holding register from device memory.

Parameters:
reference – the register reference.
Returns:
the holding register value

```
read_holding_register_bdouble(reference: int) → float
```

Read a double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_bdouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_bfloat(reference: int) → float
```

Read a float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_bfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_bitfield(reference: int, length: int) → int
```

Read a bitfield from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_holding_register_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → ByteStruct | ByteType
```

Read data from holding registers device memory.

Parameters:

- **reference** – the register reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_holding_register_double21436587(reference: int) → float
```

Read a 21436587-endian double from holding register memory.

Parameters:

reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_double21436587_list(reference: int, quantity: int) → List[float]
```

Read a list of 21436587-endian doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_double78563412(reference: int) → float
```

Read a 78563412-endian double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_double78563412_list(reference: int, quantity: int) → List[float]
```

Read a list of 78563412-endian doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_flag(reference: int) → bool
```

Read a bit from holding register memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_holding_register_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_holding_register_float2143(reference: int) → float
```

Read a 2143-endian float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_float2143_list(reference: int, quantity: int) → List[float]
```

Read a list of 2143-endian floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_float3412(reference: int) → float
```

Read a 3412-endian float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_float3412_list(reference: int, quantity: int) → List[float]
```

Read a list of 3412-endian floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_ldouble(reference: int) → float
```

Read a double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_ldouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_lfloat(reference: int) → float
```

Read a float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_lfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_nibble(reference: int) → int
```

Read a nibble from holding register memory.

Parameters:
reference – the reference to read from
Returns:

int value read from `reference`

```
read_holding_register_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint16(reference: int) → int
```

Read a signed 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint32(reference: int) → int
```

Read a signed 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint64(reference: int) → int
```

Read a signed 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint8(reference: int) → int
```

Read a signed 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint2143(reference: int) → int
```

Read a signed 2143-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint21436587(reference: int) → int
```

Read a signed 21436587-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 21436587-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 2143-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint3412(reference: int) → int
```

Read a signed 3412-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:

int value read from `reference`

```
read_holding_register_sint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 3412-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint78563412(reference: int) → int
```

Read a signed 78563412-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 78563412-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint16(reference: int) → int
```

Read a signed 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint32(reference: int) → int
```

Read a signed 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint64(reference: int) → int
```

Read a signed 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint8(reference: int) → int
```

Read a signed 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_string(reference: int, length: int) → bytes
```

Read a string from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
bytes value read from `reference`

```
read_holding_register_string_list(reference: int, quantity: int, length: int) →
List[bytes]
```

Read a list of strings from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of bytes values read from `reference`

```
read_holding_register_ubint16(reference: int) → int
```

Read an unsigned 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint32(reference: int) → int
```

Read an unsigned 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint64(reference: int) → int
```

Read an unsigned 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint8(reference: int) → int
```

Read an unsigned 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint2143(reference: int) → int
```

Read an unsigned 2143-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint21436587(reference: int) → int
```

Read an unsigned 21436587-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 21436587-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 2143-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint3412(reference: int) → int
```

Read an unsigned 3412-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 3412-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint78563412(reference: int) → int
```

Read an unsigned 78563412-endian 64-bit integer from holding register memory.

Parameters:

reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 78563412-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint16(reference: int) → int
```

Read an unsigned 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint32(reference: int) → int
```

Read an unsigned 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint64(reference: int) → int
```

Read an unsigned 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint8(reference: int) → int
```

Read an unsigned 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register(reference: int) → int
```

Read a single input register from device memory.

Parameters:
reference – the register reference.
Returns:
the input register value

```
read_input_register_bdouble(reference: int) → float
```

Read a double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_bdouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_bfloat(reference: int) → float
```

Read a float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_bfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:

list of float values read from `reference`

```
read_input_register_bitfield(reference: int, length: int) → int
```

Read a bitfield from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_input_register_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct]
| Type[ByteType]) → ByteStruct | ByteType
```

Read data from input registers from device memory.

Parameters:

- **reference** – the register reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_input_register_double21436587(reference: int) → float
```

Read a 21436587-endian double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_double21436587_list(reference: int, quantity: int) → List[float]
```

Read a list of 21436587-endian doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_double78563412(reference: int) → float
```

Read a 78563412-endian double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_double78563412_list(reference: int, quantity: int) → List[float]
```

Read a list of 78563412-endian doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_flag(reference: int) → bool
```

Read a bit from input register memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_input_register_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from input register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_input_register_float2143(reference: int) → float
```

Read a 2143-endian float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_float2143_list(reference: int, quantity: int) → List[float]
```

Read a list of 2143-endian floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_float3412(reference: int) → float
```

Read a 3412-endian float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_float3412_list(reference: int, quantity: int) → List[float]
```

Read a list of 3412-endian floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_ldouble(reference: int) → float
```

Read a double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_ldouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_lfloat(reference: int) → float
```

Read a float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_lfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_nibble(reference: int) → int
```

Read a nibble from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint16(reference: int) → int
```

Read a signed 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint32(reference: int) → int
```

Read a signed 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint64(reference: int) → int
```

Read a signed 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint8(reference: int) → int
```

Read a signed 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint2143(reference: int) → int
```

Read a signed 2143-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint21436587(reference: int) → int
```

Read a signed 21436587-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 21436587-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 2143-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint3412(reference: int) → int
```

Read a signed 3412-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 3412-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint78563412(reference: int) → int
```

Read a signed 78563412-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 78563412-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint16(reference: int) → int
```

Read a signed 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint32(reference: int) → int
```

Read a signed 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint64(reference: int) → int
```

Read a signed 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint8(reference: int) → int
```

Read a signed 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_string(reference: int, length: int) → bytes
```

Read a string from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
bytes value read from `reference`

```
read_input_register_string_list(reference: int, quantity: int, length: int) →
List[bytes]
```

Read a list of strings from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of bytes values read from `reference`

```
read_input_register_ubint16(reference: int) → int
```

Read an unsigned 16-bit integer from input register memory.

Parameters:
reference – the reference to read from

Returns:
int value read from `reference`

```
read_input_register_ubint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint32(reference: int) → int
```

Read an unsigned 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint64(reference: int) → int
```

Read an unsigned 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint8(reference: int) → int
```

Read an unsigned 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint2143(reference: int) → int
```

Read an unsigned 2143-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint21436587(reference: int) → int
```

Read an unsigned 21436587-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 21436587-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 2143-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint3412(reference: int) → int
```

Read an unsigned 3412-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 3412-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint78563412(reference: int) → int
```

Read an unsigned 78563412-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 78563412-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint16(reference: int) → int
```

Read an unsigned 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint32(reference: int) → int
```

Read an unsigned 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint64(reference: int) → int
```

Read an unsigned 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint8(reference: int) → int
```

Read an unsigned 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_multiple_coils(reference: int, quantity: int) → List[int]
```

Read multiple coils from device memory.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of coils to be read.

Returns:
a list of coil states where each element is `1` if the corresponding coil is set and `0` else.

```
read_multiple_discrete_inputs(reference: int, quantity: int) → List[int]
```

Read multiple discrete inputs from device memory.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of discrete inputs to be read.

Returns:
a list of discrete inputs where each element is `1` if the corresponding discrete input is set and `0` else.

```
read_multiple_holding_registers(reference: int, quantity: int) → List[int]
```

Read multiple holding registers from device memory.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of registers to be read.

Returns:
the holding register values

```
read_multiple_input_registers(reference: int, quantity: int) → List[int]
```

Read multiple input registers from device memory.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of registers to be read.

Returns:
the input register values

```
send_function_code_8_echo_response(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Generate an echo response using the request `pdu`.

Parameters:
pdu – the request pdu

Returns:
the response to be sent to the client

```
set_bus_character_overrun_count(count: int) → None
```

Set the bus character overrun count to `count`.

Parameters:
count – the bus character overrun count

```
set_bus_communication_error_count(count: int) → None
```

Set the bus communication error count to `count`.

Parameters:
count – the bus communication error count

```
set_bus_exception_error_count(count: int) → None
```

Set the bus exception error count to `count`.

Parameters:
count – the bus exception error count

```
set_bus_message_count(count: int) → None
```

Set the bus message count to `count`.

Parameters:
count – the bus message count

```
set_busy_count(count: int) → None
```

Set the slave busy count to `count`.

Parameters:
count – the slave busy count

```
set_busy_status(busy: bool = True) → None
```

Set busy flag which will be reported in Modbus function 11.

Parameters:
busy – if set to `True` the busy state is set to busy else not busy.

```
set_device_identifier(object_id: int | None, value: str | bytes | None) → None
```

Set a device identifier. The objects `"vendor_name"`, `"product_code"`, `"major_minor_version"`, `"vendor_url"`, `"product_name"`, `"model_name"` and `"user_application_name"` are predefined.

Set `value` to `None` to remove an entry.

Entries can be read using Modbus function code 43.14.

Parameters:

- **object_id** – the object identifier
- **value** – the value to be used

```
set_diagnostic_register(content: bytes) → None
```

Set the diagnostic register to `content`

Parameters:
content – the new content for the diagnostic register

```
set_exception_status(status: int) → None
```

Set exception status to `status`.

Parameters:
status – the new exception status

```
set_extended_memory(memory_number: int, records: Dict[int, List[int]] | None) → None
```

Set extended memory that can be read with Modbus function code 20 and written with Modbus function code 21.

If `records` is `None` the memory is removed.

Parameters:

- **memory_number** – the memory number (0x0001 to 0xFFFF)
- **records** – the records to be stored as a dict with key record numer and values list of integers

```
set_fifo_queue_content(address: int, content: List[int] | None) → None
```

Set the FIFO queue content.

Parameters:

- **address** – the address
- **content** – a list of integers or `None` to delete the queue contents

```
set_listen_only_mode(listen_only_mode_enabled: bool = True) → None
```

Set listen only mode.

Parameters:
listen_only_enabled – `True` enables the listen only mode. `False` disables it.

```
set_slave_message_count(count: int) → None
```

Set the slave message count to `count`.

Parameters:
count – the slave message count

```
set_slave_nak_count(count: int) → None
```

Set the slave nak count to `count`.

Parameters:
count – the slave nak count

```
set_slave_no_response_count(count: int) → None
```

Set the slave no response count to `count`.

Parameters:
count – the slave no response count

```
write_coil(reference: int, value: int) → None
```

Write a single coil to device memory.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written.

```
write_coil_bitfield(reference: int, value: int, length: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

- **length** – the length of the value

```
write_coil_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_coil_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct] |
Type[ByteType]) → None
```

Write data to coils device memory.

Parameters:

- **reference** – the coil reference.
- **data** – the type or instance or the decoder for the data

```
write_coil_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_coil_flag_list(reference: int, values: List[bool]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_coil_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_coil_nibble_list(reference: int, values: List[int]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_discrete_input(reference: int, value: int) → None
```

Read multiple discrete inputs from device memory

Parameters:

- **reference** – the register reference.
- **value** – the value to be written

Returns:
a list of discrete inputs where each element is `1` if the corresponding discrete input is set and `0` else.

```
write_discrete_input_bitfield(reference: int, value: int, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_discrete_input_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_discrete_input_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → None
```

Write data to discrete input device memory.

Parameters:

- **reference** – the discrete input reference.
- **data** – the type or instance or the decoder for the data

```
write_discrete_input_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_discrete_input_flag_list(reference: int, values: List[bool]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_discrete_input_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_discrete_input_nibble_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register(reference: int, value: int) → None
```

Write single holding register to device memory.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written into the corresponding register.

```
write_holding_register_bdouble(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_bdouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_bfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_bfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_bitfield(reference: int, value: int, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_holding_register_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_holding_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → None
```

Write data to holding registers device memory.

Parameters:

- **reference** – the register reference.
- **data** – the type or instance or the decoder for the data

```
write_holding_register_double21436587(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 21436587-endian double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_double21436587_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 21436587-endian doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_double78563412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 78563412-endian double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_double78563412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 78563412-endian doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_holding_register_flag_list(reference: int, values: List[bool]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_holding_register_float2143(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 2143-endian float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_float2143_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 2143-endian floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_float3412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 3412-endian float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_float3412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 3412-endian floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_ldouble(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_ldouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_lfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_lfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_nibble_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 2143-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 21436587-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 21436587-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 2143-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 3412-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 3412-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 78563412-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 78563412-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_string(reference: int, value: bytes, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a string to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bytes value to be written
- **length** – the length of the value

```
write_holding_register_string_list(reference: int, values: List[bytes], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of strings to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bytes values to be written
- **length** – the length of the value

```
write_holding_register_ubint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 2143-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 21436587-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 21436587-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 2143-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 3412-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 3412-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 78563412-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 78563412-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register(reference: int, value: int) → None
```

Write single input register to device memory.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written into the corresponding register.

```
write_input_register_bdouble(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_bdouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_bfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_bfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_bitfield(reference: int, value: int, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_input_register_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_input_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → None
```

Write data to input registers device memory.

Parameters:

- **reference** – the register reference.
- **data** – the type or instance or the decoder for the data

```
write_input_register_double21436587(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 21436587-endian double to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_double21436587_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 21436587-endian doubles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_double78563412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 78563412-endian double to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_double78563412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 78563412-endian doubles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_input_register_flag_list(reference: int, values: List[bool]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_input_register_float2143(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 2143-endian float to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_float2143_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 2143-endian floats to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_float3412(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 3412-endian float to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_float3412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 3412-endian floats to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_ldouble(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_ldouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_lfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_lfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_nibble_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sbint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sbint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sbint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sbint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sbint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sbint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sbint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sbint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 2143-endian 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 21436587-endian 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 21436587-endian 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 2143-endian 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 3412-endian 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 3412-endian 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 78563412-endian 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 78563412-endian 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_slint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_slint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_slint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_slint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_slint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_slint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_slint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_slint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_string(reference: int, value: bytes, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a string to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bytes value to be written
- **length** – the length of the value

```
write_input_register_string_list(reference: int, values: List[bytes], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of strings to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bytes values to be written
- **length** – the length of the value

```
write_input_register_ubint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ubint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ubint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ubint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ubint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ubint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ubint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ubint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_uint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 2143-endian 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_uint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 21436587-endian 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_uint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 21436587-endian 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_uint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 2143-endian 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_uint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 3412-endian 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_uint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 3412-endian 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_uint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 78563412-endian 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_uint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 78563412-endian 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ulint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ulint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ulint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ulint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ulint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ulint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ulint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ulint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_multiple_coils(reference: int, values: List[int]) → None
```

Write multiple coils to device memory.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

```
write_multiple_discrete_inputs(reference: int, values: List[int]) → None
```

Write multiple discrete inputs to device memory.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

```
write_multiple_holding_registers(reference: int, values: List[int]) → None
```

Write multiple holding registers to device memory.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

```
write_multiple_input_registers(reference: int, values: List[int]) → None
```

Write multiple input registers to device memory.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

Serial ASCII Device

```
class htf.modbus.ModbusSerialASCIIDevice(com_port: str, baud: int = 9600,
bytesize: int = 8, parity: str = 'E', stopbits: int = 1, rtscts: bool = False,
device_address: int = 1, shared_memory: bool = False, request_parser:
Type[SerialRTUModbusRequest] | Type[SerialASCIIModbusRequest] | Type[TCPModbusRequest] =
<class 'htf.modbus.protocol.SerialASCIIModbusRequest'>, response_parser:
Type[SerialRTUModbusResponse] | Type[SerialASCIIModbusResponse] | Type[TCPModbusResponse] =
<class 'htf.modbus.protocol.SerialASCIIModbusResponse'>, debuglevel: int | None = None)
```

Parameters:

- **com_port** – the com port to be used

- **baud** – the baud rate
- **bytesize** – number of bits in a byte
- **parity** – serial.PARITY_NONE, serial.PARITY_ODD or serial.PARITY_EVEN
- **stopbits** – number of stopbits (1, 1.5 or 2)
- **rtscts** – if True use flow control
- **device_address** – the device address to be used (Slave ID)
- **shared_memory** – if set to `True` a single shared memory is used
- **request_parser** – the request parser
- **response_parser** – the response parser
- **debuglevel** – the debuglevel (`None` disables debug, `1` prints requests and responses and `2` prints introspection)

```
add_communication_restart_event() → None
```

Add a communication restart event.

```
add_event(event: int) → None
```

Add an event to event log.

Parameters:
event – the event to be added.

```
add_receive_event(is_communication_error: bool = False, is_character_overrun: bool =
False, is_broadcast: bool = False, is_listen_only_mode: bool = False) → None
```

Add a receive event.

Parameters:

- **is_communication_error** – set to `True` if a communication error occurred
- **is_character_overrun** – set to `True` if a character overrun occurred
- **is_broadcast** – set to `True` if a broadcast message was received
- **is_listen_only_mode** – set to `True` if the server is currently in listen only mode

```
add_send_event(read_exception_sent: bool = False, slave_abort_exeception_sent: bool =
False, slave_busy_exception_sent: bool = False, slave_program_nak_exception_sent: bool =
False, write_timeout_occurred: bool = False) → None
```

Add a send event. The current listen only mode is determined automatically.

Parameters:

- **read_exception_sent** – set to `True` if a read exception sent occurred (exception codes 1-3)
- **slave_abort_exeception_sent** – set to `True` if a slave abort exception sent occurred (execption code 4)

- **slave_busy_exception_sent** – set to `True` if a slave busy exception sent occurred (exception codes 5-6)
- **slave_program_nak_exception_sent** – set to `True` if a slave program NAK exception sent occurred (exception code 7)
- **write_timeout_occurred** – set to `True` if a write timeout occurred

```
add_set_listen_only_mode_event() → None
```

Add a listen only mode event.

```
change_ascii_delimiter(delimiter: bytes) → None
```

Change ASCII delimiter.

Parameters:
new_lf – the new ASCII delimiter

```
clear_counters_and_diagnostic_register() → None
```

Clear counters and diagnostic register

```
clear_events() → None
```

Clear event log.

```
clear_overrun_counter_and_flag() → None
```

Clear overrun counter and reset the error flag.

```
close() → None
```

Close the device simulator and the server.

```
create_exception_response(request_pdu: RequestPDU, exception_code: str) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler to response with an exception.

Parameters:

- **request_pdu** – the request pdu
- **exception_code** – the exception code to be sent

Returns:
the response to be sent to the client

```
disable_broken_checksums() → None
```

Disable broken checksums for testing purposes.

```
enable_broken_checksums() → None
```

Enable broken checksums for testing purposes.

```
get_bus_character_overrun_count() → int
```

Get the bus character overrun count.

Returns:
the bus character overrun count

```
get_busy_count() → int
```

Return the current busy count.

Returns:
the busy count

```
get_busy_status() → bool
```

Return the current busy status.

Returns:
`True` if the device is busy else `False`

```
get_diagnostic_register() → bytes
```

Return the diagnostic register

Returns:
the content of the diagnostic register

```
get_exception_status() → int
```

Return the current exception status.

Returns:
the current exception status

```
get_extended_memory(memory_number: int) → Dict[int, List[int]] | None
```

Get extended memory.

Parameters:
memory_number – the memory number (0x0001 to 0xFFFF)
Returns:
the extended memory at `memory_number` as a dict with key record numer and values list of
integers

Return type:
list of int
Raises:
KeyError – if the `memory_number` was not found in extended memory.

```
handle_function_code_1(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 1 ``Read Coils``.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_11(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 11 `Get Comm Event Counter (Serial Line only)`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_12(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 12 `Get Comm Event Log (Serial Line only)`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_15(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 15 `Write Multiple Coils`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_16(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 16 `Write Multiple Registers`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_17(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 17 `Report Slave ID (Serial Line only)`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_2(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 2 `Read Discrete Inputs`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_20(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 20 `Read File Record`.

Parameters:
pdu – the request pdu

Returns:
the response to be sent to the client

```
handle_function_code_21(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 21 `Write File Record`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_22(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 22 `Mask Write Register`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_23(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 23 `Read/Write Multiple Registers`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_24(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 24 `Read FIFO Queue`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_3(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 3 `Read Holding Registers`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_4(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 4 `Read Input Registers`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_43(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Encapsulated Interface Transport.

Calls a handler by mei type with the name `handle_function_code_43_mei_type_<mei type>`. To add more mei types more handlers have to be added.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_43_mei_type_14(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Encapsulated Interface Transport: Read Device Identification. Basic object ids can be read.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_5(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 5 `Write Single Coil`.

Parameters:
pdu – the request pdu
Returns:

the response to be sent to the client

```
handle_function_code_6(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 6 `Write Single Register`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_7(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 7 `Read Exception Status`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.x Diagnostics (Serial Line only).

Calls a handler by sub function code like `handle_function_code_8_sub_function_<sub function code>`. To add more sub functions more handlers have to be added.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_0(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.0 `Diagnostics Send Echo Response`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_1(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.1 `Diagnostics Restart Communications Option`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_10(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.10
`Diagnostics Clear counters and diagnostic register`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_11(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.11 `Diagnostics Bus message count`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_12(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.12 `Diagnostics Bus communication error count`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_13(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.13 `Diagnostics Bus exception error count`.

Parameters:

pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_14(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.14 `Diagnostics Return slave message count`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_15(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.15 `Diagnostics Slave No Response Count`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_16(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.16 `Diagnostics Slave NAK Count`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_17(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.17 `Diagnostics Slave Busy Count`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_18(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.18 `Diagnostics Slave Character Overrun Count`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_2(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.2 `Diagnostics Return diagnostic register`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_20(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.20 `Diagnostics`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_3(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.3 `Diagnostics Change ASCII Input Delimiter`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_8_sub_function_4(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 8.4 `Diagnostics Force Listen Only Mode`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_request(adu: SerialRTUModbusRequest | SerialASCIIModbusRequest | TCPModbusRequest)
→ SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for an incoming request.

Parameters:
adu – a request adu
Returns:
the command or exception handler

```
mask_holding_register(reference: int, and_mask: int, or_mask: int) → None
```

Mask write single holding register from device memory.

Parameters:

- **reference** – the register reference
- **and_mask** – the and mask
- **or_mask** – the or mask

```
mask_multiple_holding_registers(reference: int, quantity: int, and_mask: int, or_mask:
int) → None
```

Mask write multiple holding registers from device memory.

Parameters:

- **reference** – the register reference
- **quantity** – the number of registers to be masked
- **and_mask** – the and mask
- **or_mask** – the or mask

```
read_coil(reference: int) → int
```

Read a single coil from device memory.

Parameters:
reference – the register reference.
Returns:
`1` if the coil is set and `0` else.

```
read_coil_bitfield(reference: int, length: int) → int
```

Read a bitfield from coil memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_coil_bitfield_list(reference: int, quantity: int, length: int) → List[int]
```

Read a list of bitfields from coil memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_coil_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct] |
Type[ByteType]) → ByteStruct | ByteType
```

Read data from coils device memory.

Parameters:

- **reference** – the coil reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_coil_flag(reference: int) → bool
```

Read a bit from coil memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_coil_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from coil memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_coil_nibble(reference: int) → int
```

Read a nibble from coil memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_coil_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from coil memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_discrete_input(reference: int) → int
```

Read single discrete input from device memory

Parameters:
reference – the register reference.
Returns:
`1` if the discrete input is set and `0` else.

```
read_discrete_input_bitfield(reference: int, length: int) → int
```

Read a bitfield from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_discrete_input_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_discrete_input_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct]
| Type[ByteType]) → ByteStruct | ByteType
```

Read data from discrete inputs device memory.

Parameters:

- **reference** – the discrete input reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_discrete_input_flag(reference: int) → bool
```

Read a bit from discrete input memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_discrete_input_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_discrete_input_nibble(reference: int) → int
```

Read a nibble from discrete input memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_discrete_input_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register(reference: int) → int
```

Read a single holding register from device memory.

Parameters:
reference – the register reference.
Returns:
the holding register value

```
read_holding_register_bdouble(reference: int) → float
```

Read a double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_bdouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_bfloat(reference: int) → float
```

Read a float from holding register memory.

Parameters:
reference – the reference to read from
Returns:

float value read from `reference`

```
read_holding_register_bfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_bitfield(reference: int, length: int) → int
```

Read a bitfield from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_holding_register_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → ByteStruct | ByteType
```

Read data from holding registers device memory.

Parameters:

- **reference** – the register reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_holding_register_double21436587(reference: int) → float
```

Read a 21436587-endian double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_double21436587_list(reference: int, quantity: int) → List[float]
```

Read a list of 21436587-endian doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_double78563412(reference: int) → float
```

Read a 78563412-endian double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_double78563412_list(reference: int, quantity: int) → List[float]
```

Read a list of 78563412-endian doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_flag(reference: int) → bool
```

Read a bit from holding register memory.

Parameters:
reference – the reference to read from
Returns:

bool value read from `reference`

```
read_holding_register_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_holding_register_float2143(reference: int) → float
```

Read a 2143-endian float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_float2143_list(reference: int, quantity: int) → List[float]
```

Read a list of 2143-endian floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_float3412(reference: int) → float
```

Read a 3412-endian float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_float3412_list(reference: int, quantity: int) → List[float]
```

Read a list of 3412-endian floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_ldouble(reference: int) → float
```

Read a double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_ldouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_lfloat(reference: int) → float
```

Read a float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_lfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_nibble(reference: int) → int
```

Read a nibble from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint16(reference: int) → int
```

Read a signed 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint32(reference: int) → int
```

Read a signed 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:

int value read from `reference`

```
read_holding_register_sbint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint64(reference: int) → int
```

Read a signed 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint8(reference: int) → int
```

Read a signed 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint2143(reference: int) → int
```

Read a signed 2143-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint21436587(reference: int) → int
```

Read a signed 21436587-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 21436587-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 2143-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint3412(reference: int) → int
```

Read a signed 3412-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 3412-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint78563412(reference: int) → int
```

Read a signed 78563412-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 78563412-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint16(reference: int) → int
```

Read a signed 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:

int value read from `reference`

```
read_holding_register_slint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint32(reference: int) → int
```

Read a signed 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint64(reference: int) → int
```

Read a signed 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint8(reference: int) → int
```

Read a signed 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_string(reference: int, length: int) → bytes
```

Read a string from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
bytes value read from `reference`

```
read_holding_register_string_list(reference: int, quantity: int, length: int) →
List[bytes]
```

Read a list of strings from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of bytes values read from `reference`

```
read_holding_register_ubint16(reference: int) → int
```

Read an unsigned 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint32(reference: int) → int
```

Read an unsigned 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint64(reference: int) → int
```

Read an unsigned 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint8(reference: int) → int
```

Read an unsigned 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint2143(reference: int) → int
```

Read an unsigned 2143-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint21436587(reference: int) → int
```

Read an unsigned 21436587-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 21436587-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 2143-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint3412(reference: int) → int
```

Read an unsigned 3412-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 3412-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint78563412(reference: int) → int
```

Read an unsigned 78563412-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 78563412-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint16(reference: int) → int
```

Read an unsigned 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint32(reference: int) → int
```

Read an unsigned 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:

int value read from `reference`

```
read_holding_register_ulint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint64(reference: int) → int
```

Read an unsigned 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint8(reference: int) → int
```

Read an unsigned 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register(reference: int) → int
```

Read a single input register from device memory.

Parameters:
reference – the register reference.
Returns:
the input register value

```
read_input_register_bdouble(reference: int) → float
```

Read a double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_bdouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_bfloat(reference: int) → float
```

Read a float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_bfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_bitfield(reference: int, length: int) → int
```

Read a bitfield from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_input_register_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct]
| Type[ByteType]) → ByteStruct | ByteType
```

Read data from input registers from device memory.

Parameters:

- **reference** – the register reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_input_register_double21436587(reference: int) → float
```

Read a 21436587-endian double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_double21436587_list(reference: int, quantity: int) → List[float]
```

Read a list of 21436587-endian doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_double78563412(reference: int) → float
```

Read a 78563412-endian double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_double78563412_list(reference: int, quantity: int) → List[float]
```

Read a list of 78563412-endian doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_flag(reference: int) → bool
```

Read a bit from input register memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_input_register_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_input_register_float2143(reference: int) → float
```

Read a 2143-endian float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_float2143_list(reference: int, quantity: int) → List[float]
```

Read a list of 2143-endian floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_float3412(reference: int) → float
```

Read a 3412-endian float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_float3412_list(reference: int, quantity: int) → List[float]
```

Read a list of 3412-endian floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_ldouble(reference: int) → float
```

Read a double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_ldouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_lfloat(reference: int) → float
```

Read a float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_lfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_nibble(reference: int) → int
```

Read a nibble from input register memory.

Parameters:
reference – the reference to read from
Returns:

int value read from `reference`

```
read_input_register_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint16(reference: int) → int
```

Read a signed 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint32(reference: int) → int
```

Read a signed 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint64(reference: int) → int
```

Read a signed 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint8(reference: int) → int
```

Read a signed 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint2143(reference: int) → int
```

Read a signed 2143-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint21436587(reference: int) → int
```

Read a signed 21436587-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 21436587-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 2143-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint3412(reference: int) → int
```

Read a signed 3412-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:

int value read from `reference`

```
read_input_register_sint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 3412-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint78563412(reference: int) → int
```

Read a signed 78563412-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 78563412-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint16(reference: int) → int
```

Read a signed 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint32(reference: int) → int
```

Read a signed 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint64(reference: int) → int
```

Read a signed 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint8(reference: int) → int
```

Read a signed 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_string(reference: int, length: int) → bytes
```

Read a string from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
bytes value read from `reference`

```
read_input_register_string_list(reference: int, quantity: int, length: int) →
List[bytes]
```

Read a list of strings from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of bytes values read from `reference`

```
read_input_register_ubint16(reference: int) → int
```

Read an unsigned 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint32(reference: int) → int
```

Read an unsigned 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint64(reference: int) → int
```

Read an unsigned 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint8(reference: int) → int
```

Read an unsigned 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint2143(reference: int) → int
```

Read an unsigned 2143-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint21436587(reference: int) → int
```

Read an unsigned 21436587-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 21436587-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 2143-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint3412(reference: int) → int
```

Read an unsigned 3412-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 3412-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint78563412(reference: int) → int
```

Read an unsigned 78563412-endian 64-bit integer from input register memory.

Parameters:

reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 78563412-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint16(reference: int) → int
```

Read an unsigned 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint32(reference: int) → int
```

Read an unsigned 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint64(reference: int) → int
```

Read an unsigned 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint8(reference: int) → int
```

Read an unsigned 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_multiple_coils(reference: int, quantity: int) → List[int]
```

Read multiple coils from device memory.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of coils to be read.

Returns:
a list of coil states where each element is `1` if the corresponding coil is set and `0` else.

```
read_multiple_discrete_inputs(reference: int, quantity: int) → List[int]
```

Read multiple discrete inputs from device memory.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of discrete inputs to be read.

Returns:
a list of discrete inputs where each element is `1` if the corresponding discrete input is set and `0` else.

```
read_multiple_holding_registers(reference: int, quantity: int) → List[int]
```

Read multiple holding registers from device memory.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of registers to be read.

Returns:
the holding register values

```
read_multiple_input_registers(reference: int, quantity: int) → List[int]
```

Read multiple input registers from device memory.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of registers to be read.

Returns:
the input register values

```
send_function_code_8_echo_response(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Generate an echo response using the request `pdu`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
set_bus_character_overrun_count(count: int) → None
```

Set the bus character overrun count to `count`.

Parameters:
count – the bus character overrun count

```
set_bus_communication_error_count(count: int) → None
```

Set the bus communication error count to `count`.

Parameters:
count – the bus communication error count

```
set_bus_exception_error_count(count: int) → None
```

Set the bus exception error count to `count`.

Parameters:
count – the bus exception error count

```
set_bus_message_count(count: int) → None
```

Set the bus message count to `count`.

Parameters:
count – the bus message count

```
set_busy_count(count: int) → None
```

Set the slave busy count to `count`.

Parameters:
count – the slave busy count

```
set_busy_status(busy: bool = True) → None
```

Set busy flag which will be reported in Modbus function 11.

Parameters:
busy – if set to `True` the busy state is set to busy else not busy.

```
set_device_identifier(object_id: int | None, value: str | bytes | None) → None
```

Set a device identifier. The objects `"vendor_name"`, `"product_code"`, `"major_minor_version"`, `"vendor_url"`, `"product_name"`, `"model_name"` and `"user_application_name"` are predefined.

Set `value` to `None` to remove an entry.

Entries can be read using Modbus function code 43.14.

Parameters:

- **object_id** – the object identifier
- **value** – the value to be used

```
set_diagnostic_register(content: bytes) → None
```

Set the diagnostic register to `content`

Parameters:
content – the new content for the diagnostic register

```
set_exception_status(status: int) → None
```

Set exception status to `status`.

Parameters:
status – the new exception status

```
set_extended_memory(memory_number: int, records: Dict[int, List[int]] | None) → None
```

Set extended memory that can be read with Modbus function code 20 and written with Modbus function code 21.

If `records` is `None` the memory is removed.

Parameters:

- **memory_number** – the memory number (0x0001 to 0xFFFF)
- **records** – the records to be stored as a dict with key record numer and values list of integers

```
set_fifo_queue_content(address: int, content: List[int] | None) → None
```

Set the FIFO queue content.

Parameters:

- **address** – the address
- **content** – a list of integers or `None` to delete the queue contents

```
set_listen_only_mode(listen_only_mode_enabled: bool = True) → None
```

Set listen only mode.

Parameters:
listen_only_enabled – `True` enables the listen only mode. `False` disables it.

```
set_slave_message_count(count: int) → None
```

Set the slave message count to `count`.

Parameters:
count – the slave message count

```
set_slave_nak_count(count: int) → None
```

Set the slave nak count to `count`.

Parameters:
count – the slave nak count

```
set_slave_no_response_count(count: int) → None
```

Set the slave no response count to `count`.

Parameters:
count – the slave no response count

```
write_coil(reference: int, value: int) → None
```

Write a single coil to device memory.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written.

```
write_coil_bitfield(reference: int, value: int, length: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_coil_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_coil_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct] |
Type[ByteType]) → None
```

Write data to coils device memory.

Parameters:

- **reference** – the coil reference.
- **data** – the type or instance or the decoder for the data

```
write_coil_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_coil_flag_list(reference: int, values: List[bool]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_coil_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_coil_nibble_list(reference: int, values: List[int]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_discrete_input(reference: int, value: int) → None
```

Read multiple discrete inputs from device memory

Parameters:

- **reference** – the register reference.
- **value** – the value to be written

Returns:
a list of discrete inputs where each element is `1` if the corresponding discrete input is set and `0` else.

```
write_discrete_input_bitfield(reference: int, value: int, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_discrete_input_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_discrete_input_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → None
```

Write data to discrete input device memory.

Parameters:

- **reference** – the discrete input reference.
- **data** – the type or instance or the decoder for the data

```
write_discrete_input_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_discrete_input_flag_list(reference: int, values: List[bool]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_discrete_input_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_discrete_input_nibble_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register(reference: int, value: int) → None
```

Write single holding register to device memory.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written into the corresponding register.

```
write_holding_register_bdouble(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_bdouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_bfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_bfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_bitfield(reference: int, value: int, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_holding_register_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_holding_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → None
```

Write data to holding registers device memory.

Parameters:

- **reference** – the register reference.
- **data** – the type or instance or the decoder for the data

```
write_holding_register_double21436587(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 21436587-endian double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_double21436587_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 21436587-endian doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_double78563412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 78563412-endian double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_double78563412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 78563412-endian doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_holding_register_flag_list(reference: int, values: List[bool]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_holding_register_float2143(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 2143-endian float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_float2143_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 2143-endian floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_float3412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 3412-endian float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_float3412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 3412-endian floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_ldouble(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_ldouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_lfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_lfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_nibble_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 2143-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 21436587-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 21436587-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 2143-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 3412-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 3412-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 78563412-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 78563412-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_string(reference: int, value: bytes, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a string to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bytes value to be written

- **length** – the length of the value

```
write_holding_register_string_list(reference: int, values: List[bytes], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of strings to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bytes values to be written
- **length** – the length of the value

```
write_holding_register_ubint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 2143-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 21436587-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 21436587-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 2143-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 3412-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 3412-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 78563412-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 78563412-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register(reference: int, value: int) → None
```

Write single input register to device memory.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written into the corresponding register.

```
write_input_register_bdouble(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_bdouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_bfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_bfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_bitfield(reference: int, value: int, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_input_register_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_input_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → None
```

Write data to input registers device memory.

Parameters:

- **reference** – the register reference.
- **data** – the type or instance or the decoder for the data

```
write_input_register_double21436587(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 21436587-endian double to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_double21436587_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 21436587-endian doubles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_double78563412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 78563412-endian double to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_double78563412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 78563412-endian doubles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_input_register_flag_list(reference: int, values: List[bool]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_input_register_float2143(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 2143-endian float to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_float2143_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 2143-endian floats to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_float3412(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 3412-endian float to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_float3412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 3412-endian floats to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_ldouble(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_ldouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_lfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_lfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_nibble_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sbint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sbint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sbint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sbint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sbint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sbint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sbint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sbint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 2143-endian 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 21436587-endian 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 21436587-endian 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 2143-endian 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 3412-endian 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 3412-endian 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 78563412-endian 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 78563412-endian 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_slint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_slint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_slint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_slint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_slint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_slint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_slint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_slint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_string(reference: int, value: bytes, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a string to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bytes value to be written

- **length** – the length of the value

```
write_input_register_string_list(reference: int, values: List[bytes], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of strings to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bytes values to be written
- **length** – the length of the value

```
write_input_register_ubint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ubint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ubint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ubint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ubint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ubint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ubint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ubint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_uint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 2143-endian 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_uint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 21436587-endian 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_uint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 21436587-endian 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_uint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 2143-endian 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_uint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 3412-endian 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_uint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 3412-endian 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_uint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 78563412-endian 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_uint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 78563412-endian 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ulint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ulint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ulint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ulint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ulint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ulint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ulint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ulint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_multiple_coils(reference: int, values: List[int]) → None
```

Write multiple coils to device memory.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

```
write_multiple_discrete_inputs(reference: int, values: List[int]) → None
```

Write multiple discrete inputs to device memory.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

```
write_multiple_holding_registers(reference: int, values: List[int]) → None
```

Write multiple holding registers to device memory.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

```
write_multiple_input_registers(reference: int, values: List[int]) → None
```

Write multiple input registers to device memory.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

TCP Device

```
class htf.modbus.ModbusTCPDevice(bind_address: str, port: int = 502, timeout: int | float =
1.0, shared_memory: bool = False, request_parser: Type[SerialRTUModbusRequest] |
Type[SerialASCIIModbusRequest] | Type[TCPModbusRequest] = <class
'htf.modbus.protocol.TCPModbusRequest'>, response_parser: Type[SerialRTUModbusResponse] |
```

```
Type[SerialASCIIModbusResponse] | Type[TCPModbusResponse] = <class
'htf.modbus.protocol.TCPModbusResponse'>, debuglevel: int | None = None)
```

Parameters:

- **bind_address** – the ip-address to listen on
- **port** – the port to listen on
- **timeout** – the timeout in seconds
- **shared_memory** – if set to `True` a single shared memory is used
- **request_parser** – the request parser
- **response_parser** – the response parser
- **debuglevel** – the debuglevel (`None` disables debug, `1` prints requests and responses and `2` prints introspection)

```
close() → None
```

Close the device simulator and the server.

```
create_exception_response(request_pdu: RequestPDU, exception_code: str) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler to response with an exception.

Parameters:

- **request_pdu** – the request pdu
- **exception_code** – the exception code to be sent

Returns:
the response to be sent to the client

```
get_extended_memory(memory_number: int) → Dict[int, List[int]] | None
```

Get extended memory.

Parameters:
memory_number – the memory number (0x0001 to 0xFFFF)
Returns:
the extended memory at `memory_number` as a dict with key record numer and values list of
integers

Return type:
list of int
Raises:
KeyError – if the `memory_number` was not found in extended memory.

```
handle_function_code_1(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 1 ``Read Coils``.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_15(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 15 `Write Multiple Coils`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_16(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 16 `Write Multiple Registers`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_2(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 2 `Read Discrete Inputs`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_20(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 20 `Read File Record`.

Parameters:
pdu – the request pdu

Returns:
the response to be sent to the client

```
handle_function_code_21(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 21 `Write File Record`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_22(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 22 `Mask Write Register`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_23(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 23 `Read/Write Multiple Registers`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_24(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 24 `Read FIFO Queue`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_3(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 3 `Read Holding Registers`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_4(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 4 `Read Input Registers`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_43(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Encapsulated Interface Transport.

Calls a handler by mei type with the name `handle_function_code_43_mei_type_<mei type>`. To add more mei types more handlers have to be added.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_43_mei_type_14(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Encapsulated Interface Transport: Read Device Identification. Basic object ids can be read.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_5(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 5 `Write Single Coil`.

Parameters:
pdu – the request pdu
Returns:

the response to be sent to the client

```
handle_function_code_6(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 6 `Write Single Register`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_request(adu: SerialRTUModbusRequest | SerialASCIIModbusRequest | TCPModbusRequest)
→ SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for an incoming request.

Parameters:
adu – a request adu
Returns:
the command or exception handler

```
mask_holding_register(reference: int, and_mask: int, or_mask: int) → None
```

Mask write single holding register from device memory.

Parameters:

- **reference** – the register reference
- **and_mask** – the and mask
- **or_mask** – the or mask

```
mask_multiple_holding_registers(reference: int, quantity: int, and_mask: int, or_mask:
int) → None
```

Mask write multiple holding registers from device memory.

Parameters:

- **reference** – the register reference
- **quantity** – the number of registers to be masked
- **and_mask** – the and mask
- **or_mask** – the or mask

```
read_coil(reference: int) → int
```

Read a single coil from device memory.

Parameters:
reference – the register reference.
Returns:
`1` if the coil is set and `0` else.

```
read_coil_bitfield(reference: int, length: int) → int
```

Read a bitfield from coil memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_coil_bitfield_list(reference: int, quantity: int, length: int) → List[int]
```

Read a list of bitfields from coil memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_coil_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct] |
Type[ByteType]) → ByteStruct | ByteType
```

Read data from coils device memory.

Parameters:

- **reference** – the coil reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_coil_flag(reference: int) → bool
```

Read a bit from coil memory.

Parameters:
reference – the reference to read from

Returns:
bool value read from `reference`

```
read_coil_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from coil memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_coil_nibble(reference: int) → int
```

Read a nibble from coil memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_coil_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from coil memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_discrete_input(reference: int) → int
```

Read single discrete input from device memory

Parameters:
reference – the register reference.
Returns:
`1` if the discrete input is set and `0` else.

```
read_discrete_input_bitfield(reference: int, length: int) → int
```

Read a bitfield from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_discrete_input_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_discrete_input_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct]
| Type[ByteType]) → ByteStruct | ByteType
```

Read data from discrete inputs device memory.

Parameters:

- **reference** – the discrete input reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_discrete_input_flag(reference: int) → bool
```

Read a bit from discrete input memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_discrete_input_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_discrete_input_nibble(reference: int) → int
```

Read a nibble from discrete input memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_discrete_input_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register(reference: int) → int
```

Read a single holding register from device memory.

Parameters:
reference – the register reference.
Returns:
the holding register value

```
read_holding_register_bdouble(reference: int) → float
```

Read a double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_bdouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_bfloat(reference: int) → float
```

Read a float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_bfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_bitfield(reference: int, length: int) → int
```

Read a bitfield from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_holding_register_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → ByteStruct | ByteType
```

Read data from holding registers device memory.

Parameters:

- **reference** – the register reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_holding_register_double21436587(reference: int) → float
```

Read a 21436587-endian double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_double21436587_list(reference: int, quantity: int) → List[float]
```

Read a list of 21436587-endian doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_double78563412(reference: int) → float
```

Read a 78563412-endian double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_double78563412_list(reference: int, quantity: int) → List[float]
```

Read a list of 78563412-endian doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_flag(reference: int) → bool
```

Read a bit from holding register memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_holding_register_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_holding_register_float2143(reference: int) → float
```

Read a 2143-endian float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_float2143_list(reference: int, quantity: int) → List[float]
```

Read a list of 2143-endian floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_float3412(reference: int) → float
```

Read a 3412-endian float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_float3412_list(reference: int, quantity: int) → List[float]
```

Read a list of 3412-endian floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_ldouble(reference: int) → float
```

Read a double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_ldouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_lfloat(reference: int) → float
```

Read a float from holding register memory.

Parameters:
reference – the reference to read from
Returns:

float value read from `reference`

```
read_holding_register_lfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_nibble(reference: int) → int
```

Read a nibble from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint16(reference: int) → int
```

Read a signed 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint32(reference: int) → int
```

Read a signed 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint64(reference: int) → int
```

Read a signed 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint8(reference: int) → int
```

Read a signed 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint2143(reference: int) → int
```

Read a signed 2143-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint21436587(reference: int) → int
```

Read a signed 21436587-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 21436587-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:

list of int values read from `reference`

```
read_holding_register_sint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 2143-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint3412(reference: int) → int
```

Read a signed 3412-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 3412-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint78563412(reference: int) → int
```

Read a signed 78563412-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 78563412-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint16(reference: int) → int
```

Read a signed 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint32(reference: int) → int
```

Read a signed 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint64(reference: int) → int
```

Read a signed 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint8(reference: int) → int
```

Read a signed 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_string(reference: int, length: int) → bytes
```

Read a string from holding register memory.

Parameters:

- **reference** – the reference to read from

- **length** – the length of the value

Returns:
bytes value read from `reference`

```
read_holding_register_string_list(reference: int, quantity: int, length: int) →
List[bytes]
```

Read a list of strings from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of bytes values read from `reference`

```
read_holding_register_ubint16(reference: int) → int
```

Read an unsigned 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint32(reference: int) → int
```

Read an unsigned 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint64(reference: int) → int
```

Read an unsigned 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint8(reference: int) → int
```

Read an unsigned 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint2143(reference: int) → int
```

Read an unsigned 2143-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint21436587(reference: int) → int
```

Read an unsigned 21436587-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 21436587-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 2143-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint3412(reference: int) → int
```

Read an unsigned 3412-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 3412-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint78563412(reference: int) → int
```

Read an unsigned 78563412-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 78563412-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint16(reference: int) → int
```

Read an unsigned 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint32(reference: int) → int
```

Read an unsigned 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint64(reference: int) → int
```

Read an unsigned 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint8(reference: int) → int
```

Read an unsigned 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register(reference: int) → int
```

Read a single input register from device memory.

Parameters:
reference – the register reference.
Returns:
the input register value

```
read_input_register_bdouble(reference: int) → float
```

Read a double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_bdouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_bfloat(reference: int) → float
```

Read a float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_bfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_bitfield(reference: int, length: int) → int
```

Read a bitfield from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_input_register_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct]
| Type[ByteType]) → ByteStruct | ByteType
```

Read data from input registers from device memory.

Parameters:

- **reference** – the register reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_input_register_double21436587(reference: int) → float
```

Read a 21436587-endian double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_double21436587_list(reference: int, quantity: int) → List[float]
```

Read a list of 21436587-endian doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_double78563412(reference: int) → float
```

Read a 78563412-endian double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_double78563412_list(reference: int, quantity: int) → List[float]
```

Read a list of 78563412-endian doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_flag(reference: int) → bool
```

Read a bit from input register memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_input_register_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_input_register_float2143(reference: int) → float
```

Read a 2143-endian float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_float2143_list(reference: int, quantity: int) → List[float]
```

Read a list of 2143-endian floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_float3412(reference: int) → float
```

Read a 3412-endian float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_float3412_list(reference: int, quantity: int) → List[float]
```

Read a list of 3412-endian floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_ldouble(reference: int) → float
```

Read a double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_ldouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_lfloat(reference: int) → float
```

Read a float from input register memory.

Parameters:
reference – the reference to read from
Returns:

float value read from `reference`

```
read_input_register_lfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_nibble(reference: int) → int
```

Read a nibble from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint16(reference: int) → int
```

Read a signed 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint32(reference: int) → int
```

Read a signed 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint64(reference: int) → int
```

Read a signed 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint8(reference: int) → int
```

Read a signed 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint2143(reference: int) → int
```

Read a signed 2143-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint21436587(reference: int) → int
```

Read a signed 21436587-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 21436587-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:

list of int values read from `reference`

```
read_input_register_sint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 2143-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint3412(reference: int) → int
```

Read a signed 3412-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 3412-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint78563412(reference: int) → int
```

Read a signed 78563412-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 78563412-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint16(reference: int) → int
```

Read a signed 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint32(reference: int) → int
```

Read a signed 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint64(reference: int) → int
```

Read a signed 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint8(reference: int) → int
```

Read a signed 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_string(reference: int, length: int) → bytes
```

Read a string from input register memory.

Parameters:

- **reference** – the reference to read from

- **length** – the length of the value

Returns:
bytes value read from `reference`

```
read_input_register_string_list(reference: int, quantity: int, length: int) →
List[bytes]
```

Read a list of strings from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of bytes values read from `reference`

```
read_input_register_ubint16(reference: int) → int
```

Read an unsigned 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint32(reference: int) → int
```

Read an unsigned 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint64(reference: int) → int
```

Read an unsigned 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint8(reference: int) → int
```

Read an unsigned 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint2143(reference: int) → int
```

Read an unsigned 2143-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint21436587(reference: int) → int
```

Read an unsigned 21436587-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 21436587-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 2143-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint3412(reference: int) → int
```

Read an unsigned 3412-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 3412-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint78563412(reference: int) → int
```

Read an unsigned 78563412-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 78563412-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint16(reference: int) → int
```

Read an unsigned 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint32(reference: int) → int
```

Read an unsigned 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint64(reference: int) → int
```

Read an unsigned 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint8(reference: int) → int
```

Read an unsigned 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_multiple_coils(reference: int, quantity: int) → List[int]
```

Read multiple coils from device memory.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of coils to be read.

Returns:
a list of coil states where each element is `1` if the corresponding coil is set and `0` else.

```
read_multiple_discrete_inputs(reference: int, quantity: int) → List[int]
```

Read multiple discrete inputs from device memory.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of discrete inputs to be read.

Returns:
a list of discrete inputs where each element is `1` if the corresponding discrete input is set and `0` else.

```
read_multiple_holding_registers(reference: int, quantity: int) → List[int]
```

Read multiple holding registers from device memory.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of registers to be read.

Returns:
the holding register values

```
read_multiple_input_registers(reference: int, quantity: int) → List[int]
```

Read multiple input registers from device memory.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of registers to be read.

Returns:
the input register values

```
set_device_identifier(object_id: int | None, value: str | bytes | None) → None
```

Set a device identifier. The objects `"vendor_name"`, `"product_code"`, `"major_minor_version"`, `"vendor_url"`, `"product_name"`, `"model_name"` and `"user_application_name"` are predefined.

Set `value` to `None` to remove an entry.

Entries can be read using Modbus function code 43.14.

Parameters:

- **object_id** – the object identifier
- **value** – the value to be used

```
set_extended_memory(memory_number: int, records: Dict[int, List[int]] | None) → None
```

Set extended memory that can be read with Modbus function code 20 and written with Modbus function code 21.

If `records` is `None` the memory is removed.

Parameters:

- **memory_number** – the memory number (0x0001 to 0xFFFF)
- **records** – the records to be stored as a dict with key record numer and values list of integers

```
set_fifo_queue_content(address: int, content: List[int] | None) → None
```

Set the FIFO queue content.

Parameters:

- **address** – the address
- **content** – a list of integers or `None` to delete the queue contents

```
write_coil(reference: int, value: int) → None
```

Write a single coil to device memory.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written.

```
write_coil_bitfield(reference: int, value: int, length: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_coil_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_coil_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct] |
Type[ByteType]) → None
```

Write data to coils device memory.

Parameters:

- **reference** – the coil reference.

- **data** – the type or instance or the decoder for the data

```
write_coil_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_coil_flag_list(reference: int, values: List[bool]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_coil_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_coil_nibble_list(reference: int, values: List[int]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_discrete_input(reference: int, value: int) → None
```

Read multiple discrete inputs from device memory

Parameters:

- **reference** – the register reference.

- **value** – the value to be written

Returns:
a list of discrete inputs where each element is `1` if the corresponding discrete input is set and `0` else.

```
write_discrete_input_bitfield(reference: int, value: int, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_discrete_input_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_discrete_input_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → None
```

Write data to discrete input device memory.

Parameters:

- **reference** – the discrete input reference.
- **data** – the type or instance or the decoder for the data

```
write_discrete_input_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_discrete_input_flag_list(reference: int, values: List[bool]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_discrete_input_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_discrete_input_nibble_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register(reference: int, value: int) → None
```

Write single holding register to device memory.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written into the corresponding register.

```
write_holding_register_bdouble(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_bdouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_bfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_bfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_bitfield(reference: int, value: int, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_holding_register_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to holding register memory.

Parameters:

- **reference** – the reference to write to

- **values** – a list of int values to be written
- **length** – the length of the value

```
write_holding_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → None
```

Write data to holding registers device memory.

Parameters:

- **reference** – the register reference.
- **data** – the type or instance or the decoder for the data

```
write_holding_register_double21436587(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 21436587-endian double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_double21436587_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 21436587-endian doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_double78563412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 78563412-endian double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_double78563412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 78563412-endian doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_holding_register_flag_list(reference: int, values: List[bool]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_holding_register_float2143(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 2143-endian float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_float2143_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 2143-endian floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_float3412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 3412-endian float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_float3412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 3412-endian floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_ldouble(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_ldouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_lfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_lfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_nibble_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 2143-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 21436587-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 21436587-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 2143-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 3412-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 3412-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 78563412-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 78563412-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_string(reference: int, value: bytes, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a string to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bytes value to be written
- **length** – the length of the value

```
write_holding_register_string_list(reference: int, values: List[bytes], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of strings to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bytes values to be written
- **length** – the length of the value

```
write_holding_register_ubint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 2143-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 21436587-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 21436587-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 2143-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 3412-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 3412-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 78563412-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 78563412-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register(reference: int, value: int) → None
```

Write single input register to device memory.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written into the corresponding register.

```
write_input_register_bdouble(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_bdouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_bfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_bfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_bitfield(reference: int, value: int, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_input_register_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to input register memory.

Parameters:

- **reference** – the reference to write to

- **values** – a list of int values to be written
- **length** – the length of the value

```
write_input_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → None
```

Write data to input registers device memory.

Parameters:

- **reference** – the register reference.
- **data** – the type or instance or the decoder for the data

```
write_input_register_double21436587(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 21436587-endian double to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_double21436587_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 21436587-endian doubles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_double78563412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 78563412-endian double to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_double78563412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 78563412-endian doubles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_input_register_flag_list(reference: int, values: List[bool]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_input_register_float2143(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 2143-endian float to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_float2143_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 2143-endian floats to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_float3412(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 3412-endian float to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_float3412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 3412-endian floats to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_ldouble(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_ldouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_lfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_lfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_nibble_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sbint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sbint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sbint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sbint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sbint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sbint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sbint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sbint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 2143-endian 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 21436587-endian 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 21436587-endian 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 2143-endian 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 3412-endian 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 3412-endian 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 78563412-endian 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 78563412-endian 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_slint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_slint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_slint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_slint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_slint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_slint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_slint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_slint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_string(reference: int, value: bytes, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a string to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bytes value to be written
- **length** – the length of the value

```
write_input_register_string_list(reference: int, values: List[bytes], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of strings to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bytes values to be written
- **length** – the length of the value

```
write_input_register_ubint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ubint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ubint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ubint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ubint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ubint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ubint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ubint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_uint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 2143-endian 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_uint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 21436587-endian 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_uint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 21436587-endian 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_uint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 2143-endian 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_uint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 3412-endian 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_uint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 3412-endian 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_uint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 78563412-endian 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_uint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 78563412-endian 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ulint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ulint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ulint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ulint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ulint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ulint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ulint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ulint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_multiple_coils(reference: int, values: List[int]) → None
```

Write multiple coils to device memory.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

```
write_multiple_discrete_inputs(reference: int, values: List[int]) → None
```

Write multiple discrete inputs to device memory.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

```
write_multiple_holding_registers(reference: int, values: List[int]) → None
```

Write multiple holding registers to device memory.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

```
write_multiple_input_registers(reference: int, values: List[int]) → None
```

Write multiple input registers to device memory.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

UDP Device

```
class htf.modbus.ModbusUDPDevice(bind_address: str, port: int = 502, shared_memory: bool =
False, request_parser: Type[SerialRTUModbusRequest] | Type[SerialASCIIModbusRequest] |
Type[TCPModbusRequest] = <class 'htf.modbus.protocol.TCPModbusRequest'>, response_parser:
Type[SerialRTUModbusResponse] | Type[SerialASCIIModbusResponse] | Type[TCPModbusResponse] =
<class 'htf.modbus.protocol.TCPModbusResponse'>, debuglevel: int | None = None)
```

Parameters:

- **bind_address** – the ip-address to listen on
- **port** – the port to listen on
- **shared_memory** – if set to `True` a single shared memory is used
- **request_parser** – the request parser
- **response_parser** – the response parser
- **debuglevel** – the debuglevel (`None` disables debug, `1` prints requests and responses and `2` prints introspection)

```
close() → None
```

Close the device simulator and the server.

```
create_exception_response(request_pdu: RequestPDU, exception_code: str) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler to response with an exception.

Parameters:

- **request_pdu** – the request pdu
- **exception_code** – the exception code to be sent

Returns:
the response to be sent to the client

```
get_extended_memory(memory_number: int) → Dict[int, List[int]] | None
```

Get extended memory.

Parameters:
memory_number – the memory number (0x0001 to 0xFFFF)
Returns:
the extended memory at `memory_number` as a dict with key record numer and values list of integers

Return type:
list of int
Raises:
KeyError – if the `memory_number` was not found in extended memory.

```
handle_function_code_1(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 1 ``Read Coils``.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_15(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 15 `Write Multiple Coils`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_16(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 16 `Write Multiple Registers`.

Parameters:

pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_2(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 2 `Read Discrete Inputs`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_20(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 20 `Read File Record`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_21(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 21 `Write File Record`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_22(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 22 `Mask Write Register`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_23(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 23 `Read/Write Multiple Registers`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_24(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 24 `Read FIFO Queue`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_3(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 3 `Read Holding Registers`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_4(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 4 `Read Input Registers`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_43(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Encapsulated Interface Transport.

Calls a handler by mei type with the name `handle_function_code_43_mei_type_<mei type>`. To add more mei types more handlers have to be added.

Parameters:
pdu – the request pdu
Returns:

the response to be sent to the client

```
handle_function_code_43_mei_type_14(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Encapsulated Interface Transport: Read Device Identification. Basic object ids can be read.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_5(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 5 `Write Single Coil`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_6(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 6 `Write Single Register`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_request(adu: SerialRTUModbusRequest | SerialASCIIModbusRequest | TCPModbusRequest)
→ SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for an incoming request.

Parameters:
adu – a request adu
Returns:
the command or exception handler

```
mask_holding_register(reference: int, and_mask: int, or_mask: int) → None
```

Mask write single holding register from device memory.

Parameters:

- **reference** – the register reference
- **and_mask** – the and mask
- **or_mask** – the or mask

```
mask_multiple_holding_registers(reference: int, quantity: int, and_mask: int, or_mask: int) → None
```

Mask write multiple holding registers from device memory.

Parameters:

- **reference** – the register reference
- **quantity** – the number of registers to be masked
- **and_mask** – the and mask
- **or_mask** – the or mask

```
read_coil(reference: int) → int
```

Read a single coil from device memory.

Parameters:
reference – the register reference.
Returns:
`1` if the coil is set and `0` else.

```
read_coil_bitfield(reference: int, length: int) → int
```

Read a bitfield from coil memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_coil_bitfield_list(reference: int, quantity: int, length: int) → List[int]
```

Read a list of bitfields from coil memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:

list of int values read from `reference`

```
read_coil_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct] |
Type[ByteType]) → ByteStruct | ByteType
```

Read data from coils device memory.

Parameters:

- **reference** – the coil reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_coil_flag(reference: int) → bool
```

Read a bit from coil memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_coil_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from coil memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_coil_nibble(reference: int) → int
```

Read a nibble from coil memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_coil_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from coil memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_discrete_input(reference: int) → int
```

Read single discrete input from device memory

Parameters:
reference – the register reference.
Returns:
`1` if the discrete input is set and `0` else.

```
read_discrete_input_bitfield(reference: int, length: int) → int
```

Read a bitfield from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_discrete_input_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_discrete_input_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct]
| Type[ByteType]) → ByteStruct | ByteType
```

Read data from discrete inputs device memory.

Parameters:

- **reference** – the discrete input reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_discrete_input_flag(reference: int) → bool
```

Read a bit from discrete input memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_discrete_input_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_discrete_input_nibble(reference: int) → int
```

Read a nibble from discrete input memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_discrete_input_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register(reference: int) → int
```

Read a single holding register from device memory.

Parameters:
reference – the register reference.
Returns:
the holding register value

```
read_holding_register_bdouble(reference: int) → float
```

Read a double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_bdouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_bfloat(reference: int) → float
```

Read a float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_bfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_bitfield(reference: int, length: int) → int
```

Read a bitfield from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_holding_register_bitfield_list(reference: int, quantity: int, length: int) → List[int]
```

Read a list of bitfields from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct] | Type[ByteType]) → ByteStruct | ByteType
```

Read data from holding registers device memory.

Parameters:

- **reference** – the register reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_holding_register_double21436587(reference: int) → float
```

Read a 21436587-endian double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_double21436587_list(reference: int, quantity: int) → List[float]
```

Read a list of 21436587-endian doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_double78563412(reference: int) → float
```

Read a 78563412-endian double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_double78563412_list(reference: int, quantity: int) → List[float]
```

Read a list of 78563412-endian doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_flag(reference: int) → bool
```

Read a bit from holding register memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_holding_register_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_holding_register_float2143(reference: int) → float
```

Read a 2143-endian float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_float2143_list(reference: int, quantity: int) → List[float]
```

Read a list of 2143-endian floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_float3412(reference: int) → float
```

Read a 3412-endian float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_float3412_list(reference: int, quantity: int) → List[float]
```

Read a list of 3412-endian floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_ldouble(reference: int) → float
```

Read a double from holding register memory.

Parameters:
reference – the reference to read from
Returns:

float value read from `reference`

```
read_holding_register_ldouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_lfloat(reference: int) → float
```

Read a float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_lfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_nibble(reference: int) → int
```

Read a nibble from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint16(reference: int) → int
```

Read a signed 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint32(reference: int) → int
```

Read a signed 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint64(reference: int) → int
```

Read a signed 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint8(reference: int) → int
```

Read a signed 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint2143(reference: int) → int
```

Read a signed 2143-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:

int value read from `reference`

```
read_holding_register_sint21436587(reference: int) → int
```

Read a signed 21436587-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 21436587-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 2143-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint3412(reference: int) → int
```

Read a signed 3412-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 3412-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint78563412(reference: int) → int
```

Read a signed 78563412-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 78563412-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint16(reference: int) → int
```

Read a signed 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint32(reference: int) → int
```

Read a signed 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint64(reference: int) → int
```

Read a signed 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint8(reference: int) → int
```

Read a signed 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:

int value read from `reference`

```
read_holding_register_slint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_string(reference: int, length: int) → bytes
```

Read a string from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
bytes value read from `reference`

```
read_holding_register_string_list(reference: int, quantity: int, length: int) →
List[bytes]
```

Read a list of strings from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of bytes values read from `reference`

```
read_holding_register_ubint16(reference: int) → int
```

Read an unsigned 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint32(reference: int) → int
```

Read an unsigned 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint64(reference: int) → int
```

Read an unsigned 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint8(reference: int) → int
```

Read an unsigned 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint2143(reference: int) → int
```

Read an unsigned 2143-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint21436587(reference: int) → int
```

Read an unsigned 21436587-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 21436587-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 2143-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint3412(reference: int) → int
```

Read an unsigned 3412-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 3412-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint78563412(reference: int) → int
```

Read an unsigned 78563412-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 78563412-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint16(reference: int) → int
```

Read an unsigned 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint32(reference: int) → int
```

Read an unsigned 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint64(reference: int) → int
```

Read an unsigned 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint8(reference: int) → int
```

Read an unsigned 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register(reference: int) → int
```

Read a single input register from device memory.

Parameters:
reference – the register reference.
Returns:
the input register value

```
read_input_register_bdouble(reference: int) → float
```

Read a double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_bdouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_bfloat(reference: int) → float
```

Read a float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_bfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_bitfield(reference: int, length: int) → int
```

Read a bitfield from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_input_register_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct]
| Type[ByteType]) → ByteStruct | ByteType
```

Read data from input registers from device memory.

Parameters:

- **reference** – the register reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_input_register_double21436587(reference: int) → float
```

Read a 21436587-endian double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_double21436587_list(reference: int, quantity: int) → List[float]
```

Read a list of 21436587-endian doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_double78563412(reference: int) → float
```

Read a 78563412-endian double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_double78563412_list(reference: int, quantity: int) → List[float]
```

Read a list of 78563412-endian doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_flag(reference: int) → bool
```

Read a bit from input register memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_input_register_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_input_register_float2143(reference: int) → float
```

Read a 2143-endian float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_float2143_list(reference: int, quantity: int) → List[float]
```

Read a list of 2143-endian floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_float3412(reference: int) → float
```

Read a 3412-endian float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_float3412_list(reference: int, quantity: int) → List[float]
```

Read a list of 3412-endian floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_ldouble(reference: int) → float
```

Read a double from input register memory.

Parameters:
reference – the reference to read from
Returns:

float value read from `reference`

```
read_input_register_ldouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_lfloat(reference: int) → float
```

Read a float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_lfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_nibble(reference: int) → int
```

Read a nibble from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint16(reference: int) → int
```

Read a signed 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint32(reference: int) → int
```

Read a signed 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint64(reference: int) → int
```

Read a signed 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint8(reference: int) → int
```

Read a signed 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint2143(reference: int) → int
```

Read a signed 2143-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:

int value read from `reference`

```
read_input_register_sint21436587(reference: int) → int
```

Read a signed 21436587-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 21436587-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 2143-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint3412(reference: int) → int
```

Read a signed 3412-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 3412-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint78563412(reference: int) → int
```

Read a signed 78563412-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 78563412-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint16(reference: int) → int
```

Read a signed 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint32(reference: int) → int
```

Read a signed 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint64(reference: int) → int
```

Read a signed 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint8(reference: int) → int
```

Read a signed 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:

int value read from `reference`

```
read_input_register_slint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_string(reference: int, length: int) → bytes
```

Read a string from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
bytes value read from `reference`

```
read_input_register_string_list(reference: int, quantity: int, length: int) →
List[bytes]
```

Read a list of strings from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of bytes values read from `reference`

```
read_input_register_ubint16(reference: int) → int
```

Read an unsigned 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint32(reference: int) → int
```

Read an unsigned 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint64(reference: int) → int
```

Read an unsigned 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint8(reference: int) → int
```

Read an unsigned 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint2143(reference: int) → int
```

Read an unsigned 2143-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint21436587(reference: int) → int
```

Read an unsigned 21436587-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 21436587-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 2143-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint3412(reference: int) → int
```

Read an unsigned 3412-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 3412-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint78563412(reference: int) → int
```

Read an unsigned 78563412-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 78563412-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint16(reference: int) → int
```

Read an unsigned 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint32(reference: int) → int
```

Read an unsigned 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint64(reference: int) → int
```

Read an unsigned 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint8(reference: int) → int
```

Read an unsigned 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_multiple_coils(reference: int, quantity: int) → List[int]
```

Read multiple coils from device memory.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of coils to be read.

Returns:
a list of coil states where each element is `1` if the corresponding coil is set and `0` else.

```
read_multiple_discrete_inputs(reference: int, quantity: int) → List[int]
```

Read multiple discrete inputs from device memory.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of discrete inputs to be read.

Returns:
a list of discrete inputs where each element is `1` if the corresponding discrete input is set and `0` else.

```
read_multiple_holding_registers(reference: int, quantity: int) → List[int]
```

Read multiple holding registers from device memory.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of registers to be read.

Returns:
the holding register values

```
read_multiple_input_registers(reference: int, quantity: int) → List[int]
```

Read multiple input registers from device memory.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of registers to be read.

Returns:
the input register values

```
set_device_identifier(object_id: int | None, value: str | bytes | None) → None
```

Set a device identifier. The objects `"vendor_name"`, `"product_code"`, `"major_minor_version"`, `"vendor_url"`, `"product_name"`, `"model_name"` and `"user_application_name"` are predefined.

Set `value` to `None` to remove an entry.

Entries can be read using Modbus function code 43.14.

Parameters:

- **object_id** – the object identifier
- **value** – the value to be used

```
set_extended_memory(memory_number: int, records: Dict[int, List[int]] | None) → None
```

Set extended memory that can be read with Modbus function code 20 and written with Modbus function code 21.

If `records` is `None` the memory is removed.

Parameters:

- **memory_number** – the memory number (0x0001 to 0xFFFF)
- **records** – the records to be stored as a dict with key record numer and values list of integers

```
set_fifo_queue_content(address: int, content: List[int] | None) → None
```

Set the FIFO queue content.

Parameters:

- **address** – the address
- **content** – a list of integers or `None` to delete the queue contents

```
write_coil(reference: int, value: int) → None
```

Write a single coil to device memory.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written.

```
write_coil_bitfield(reference: int, value: int, length: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_coil_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_coil_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct] |
Type[ByteType]) → None
```

Write data to coils device memory.

Parameters:

- **reference** – the coil reference.
- **data** – the type or instance or the decoder for the data

```
write_coil_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_coil_flag_list(reference: int, values: List[bool]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_coil_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to coil memory.

Parameters:

- **reference** – the reference to write to

- **value** – the int value to be written

```
write_coil_nibble_list(reference: int, values: List[int]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_discrete_input(reference: int, value: int) → None
```

Read multiple discrete inputs from device memory

Parameters:

- **reference** – the register reference.
- **value** – the value to be written

Returns:
a list of discrete inputs where each element is `1` if the corresponding discrete input is set and `0` else.

```
write_discrete_input_bitfield(reference: int, value: int, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_discrete_input_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_discrete_input_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → None
```

Write data to discrete input device memory.

Parameters:

- **reference** – the discrete input reference.
- **data** – the type or instance or the decoder for the data

```
write_discrete_input_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_discrete_input_flag_list(reference: int, values: List[bool]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_discrete_input_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_discrete_input_nibble_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register(reference: int, value: int) → None
```

Write single holding register to device memory.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written into the corresponding register.

```
write_holding_register_bdouble(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_bdouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_bfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_bfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_bitfield(reference: int, value: int, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_holding_register_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_holding_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → None
```

Write data to holding registers device memory.

Parameters:

- **reference** – the register reference.
- **data** – the type or instance or the decoder for the data

```
write_holding_register_double21436587(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 21436587-endian double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_double21436587_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 21436587-endian doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_double78563412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 78563412-endian double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_double78563412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 78563412-endian doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_holding_register_flag_list(reference: int, values: List[bool]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_holding_register_float2143(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 2143-endian float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_float2143_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 2143-endian floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_float3412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 3412-endian float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_float3412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 3412-endian floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_ldouble(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_ldouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_lfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_lfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_nibble_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 2143-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 21436587-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 21436587-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 2143-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 3412-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 3412-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 78563412-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 78563412-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_string(reference: int, value: bytes, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a string to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bytes value to be written
- **length** – the length of the value

```
write_holding_register_string_list(reference: int, values: List[bytes], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of strings to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bytes values to be written
- **length** – the length of the value

```
write_holding_register_ubint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 2143-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 21436587-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 21436587-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 2143-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 3412-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 3412-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 78563412-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 78563412-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register(reference: int, value: int) → None
```

Write single input register to device memory.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written into the corresponding register.

```
write_input_register_bdouble(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_bdouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_bfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_bfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_bitfield(reference: int, value: int, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_input_register_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_input_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → None
```

Write data to input registers device memory.

Parameters:

- **reference** – the register reference.
- **data** – the type or instance or the decoder for the data

```
write_input_register_double21436587(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 21436587-endian double to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_double21436587_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 21436587-endian doubles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_double78563412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 78563412-endian double to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_double78563412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 78563412-endian doubles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_input_register_flag_list(reference: int, values: List[bool]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_input_register_float2143(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 2143-endian float to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_float2143_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 2143-endian floats to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_float3412(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 3412-endian float to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_float3412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 3412-endian floats to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_ldouble(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_ldouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_lfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_lfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_nibble_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sbint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sbint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sbint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sbint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sbint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sbint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sbint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sbint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 2143-endian 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 21436587-endian 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 21436587-endian 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 2143-endian 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 3412-endian 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 3412-endian 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 78563412-endian 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 78563412-endian 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_slint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_slint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_slint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_slint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_slint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_slint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_slint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_slint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_string(reference: int, value: bytes, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a string to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bytes value to be written
- **length** – the length of the value

```
write_input_register_string_list(reference: int, values: List[bytes], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of strings to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bytes values to be written
- **length** – the length of the value

```
write_input_register_ubint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ubint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ubint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ubint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ubint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ubint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ubint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ubint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_uint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 2143-endian 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_uint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 21436587-endian 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_uint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 21436587-endian 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_uint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 2143-endian 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_uint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 3412-endian 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_uint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 3412-endian 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_uint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 78563412-endian 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_uint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 78563412-endian 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ulint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ulint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ulint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ulint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ulint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ulint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ulint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ulint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_multiple_coils(reference: int, values: List[int]) → None
```

Write multiple coils to device memory.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

```
write_multiple_discrete_inputs(reference: int, values: List[int]) → None
```

Write multiple discrete inputs to device memory.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

```
write_multiple_holding_registers(reference: int, values: List[int]) → None
```

Write multiple holding registers to device memory.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

```
write_multiple_input_registers(reference: int, values: List[int]) → None
```

Write multiple input registers to device memory.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

TLS Device

```
class htf.modbus.ModbusTLSDevice(bind_address: str, port: int = 502, shared_memory: bool =
False, certfile: str | None = None, keyfile: str | None = None, password: str | None = None,
request_parser: Type[SerialRTUModbusRequest] | Type[SerialASCIIModbusRequest] |
Type[TCPModbusRequest] = <class 'htf.modbus.protocol.TCPModbusRequest'>, response_parser:
Type[SerialRTUModbusResponse] | Type[SerialASCIIModbusResponse] | Type[TCPModbusResponse] =
<class 'htf.modbus.protocol.TCPModbusResponse'>, debuglevel: int | None = None)
```

Parameters:

- **bind_address** – the ip-address to listen on
- **port** – the port to listen on

- **shared_memory** – if set to `True` a single shared memory is used
- **callback** – the callback to handle requests
- **certfile** – path to TLS certificate
- **keyfile** – path to TLS keyfile
- **password** – TLS password
- **request_parser** – the request parser
- **response_parser** – the response parser
- **debuglevel** – the debuglevel (`None` disables debug, `1` prints requests and responses and `2` prints introspection)

```
close() → None
```

Close the device simulator and the server.

```
create_exception_response(request_pdu: RequestPDU, exception_code: str) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler to response with an exception.

Parameters:

- **request_pdu** – the request pdu
- **exception_code** – the exception code to be sent

Returns:
the response to be sent to the client

```
get_extended_memory(memory_number: int) → Dict[int, List[int]] | None
```

Get extended memory.

Parameters:
memory_number – the memory number (0x0001 to 0xFFFF)
Returns:
the extended memory at `memory_number` as a dict with key record numer and values list of
integers

Return type:
list of int
Raises:
KeyError – if the `memory_number` was not found in extended memory.

```
handle_function_code_1(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 1 ``` ``Read Coils`.``.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_15(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 15 `Write Multiple Coils`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_16(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 16 `Write Multiple Registers`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_2(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 2 `Read Discrete Inputs`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_20(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 20 `Read File Record`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_21(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 21 `Write File Record`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_22(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 22 `Mask Write Register`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_23(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 23 `Read/Write Multiple Registers`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_24(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 24 `Read FIFO Queue`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_3(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 3 `Read Holding Registers`.

Parameters:
pdu – the request pdu

Returns:
the response to be sent to the client

```
handle_function_code_4(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 4 `Read Input Registers`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_43(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Encapsulated Interface Transport.

Calls a handler by mei type with the name `handle_function_code_43_mei_type_<mei type>`. To add more mei types more handlers have to be added.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_43_mei_type_14(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Encapsulated Interface Transport: Read Device Identification. Basic object ids can be read.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_5(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 5 `Write Single Coil`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_function_code_6(pdu: RequestPDU) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for function code 6 `Write Single Register`.

Parameters:
pdu – the request pdu
Returns:
the response to be sent to the client

```
handle_request(adu: SerialRTUModbusRequest | SerialASCIIModbusRequest | TCPModbusRequest)
→ SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Request handler for an incoming request.

Parameters:
adu – a request adu
Returns:
the command or exception handler

```
mask_holding_register(reference: int, and_mask: int, or_mask: int) → None
```

Mask write single holding register from device memory.

Parameters:

- **reference** – the register reference
- **and_mask** – the and mask
- **or_mask** – the or mask

```
mask_multiple_holding_registers(reference: int, quantity: int, and_mask: int, or_mask:
int) → None
```

Mask write multiple holding registers from device memory.

Parameters:

- **reference** – the register reference
- **quantity** – the number of registers to be masked
- **and_mask** – the and mask
- **or_mask** – the or mask

```
read_coil(reference: int) → int
```

Read a single coil from device memory.

Parameters:

reference – the register reference.
Returns:
`1` if the coil is set and `0` else.

```
read_coil_bitfield(reference: int, length: int) → int
```

Read a bitfield from coil memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_coil_bitfield_list(reference: int, quantity: int, length: int) → List[int]
```

Read a list of bitfields from coil memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_coil_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct] |
Type[ByteType]) → ByteStruct | ByteType
```

Read data from coils device memory.

Parameters:

- **reference** – the coil reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_coil_flag(reference: int) → bool
```

Read a bit from coil memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_coil_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from coil memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_coil_nibble(reference: int) → int
```

Read a nibble from coil memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_coil_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from coil memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_discrete_input(reference: int) → int
```

Read single discrete input from device memory

Parameters:
reference – the register reference.
Returns:
`1` if the discrete input is set and `0` else.

```
read_discrete_input_bitfield(reference: int, length: int) → int
```

Read a bitfield from discrete input memory.

Parameters:

- **reference** – the reference to read from

- **length** – the length of the value

Returns:
int value read from `reference`

```
read_discrete_input_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_discrete_input_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct]
| Type[ByteType]) → ByteStruct | ByteType
```

Read data from discrete inputs device memory.

Parameters:

- **reference** – the discrete input reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_discrete_input_flag(reference: int) → bool
```

Read a bit from discrete input memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_discrete_input_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_discrete_input_nibble(reference: int) → int
```

Read a nibble from discrete input memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_discrete_input_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from discrete input memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register(reference: int) → int
```

Read a single holding register from device memory.

Parameters:
reference – the register reference.
Returns:
the holding register value

```
read_holding_register_bdouble(reference: int) → float
```

Read a double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_bdouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:

list of float values read from `reference`

```
read_holding_register_bfloat(reference: int) → float
```

Read a float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_bfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_bitfield(reference: int, length: int) → int
```

Read a bitfield from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_holding_register_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → ByteStruct | ByteType
```

Read data from holding registers device memory.

Parameters:

- **reference** – the register reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_holding_register_double21436587(reference: int) → float
```

Read a 21436587-endian double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_double21436587_list(reference: int, quantity: int) → List[float]
```

Read a list of 21436587-endian doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_double78563412(reference: int) → float
```

Read a 78563412-endian double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_double78563412_list(reference: int, quantity: int) → List[float]
```

Read a list of 78563412-endian doubles from holding register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_flag(reference: int) → bool
```

Read a bit from holding register memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_holding_register_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_holding_register_float2143(reference: int) → float
```

Read a 2143-endian float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_float2143_list(reference: int, quantity: int) → List[float]
```

Read a list of 2143-endian floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_float3412(reference: int) → float
```

Read a 3412-endian float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_float3412_list(reference: int, quantity: int) → List[float]
```

Read a list of 3412-endian floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_ldouble(reference: int) → float
```

Read a double from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_ldouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_lfloat(reference: int) → float
```

Read a float from holding register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_holding_register_lfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_holding_register_nibble(reference: int) → int
```

Read a nibble from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint16(reference: int) → int
```

Read a signed 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint32(reference: int) → int
```

Read a signed 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint64(reference: int) → int
```

Read a signed 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sbint8(reference: int) → int
```

Read a signed 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sbint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint2143(reference: int) → int
```

Read a signed 2143-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint21436587(reference: int) → int
```

Read a signed 21436587-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 21436587-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 2143-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint3412(reference: int) → int
```

Read a signed 3412-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 3412-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_sint78563412(reference: int) → int
```

Read a signed 78563412-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_sint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 78563412-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint16(reference: int) → int
```

Read a signed 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint32(reference: int) → int
```

Read a signed 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint64(reference: int) → int
```

Read a signed 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_slint8(reference: int) → int
```

Read a signed 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_slint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_string(reference: int, length: int) → bytes
```

Read a string from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
bytes value read from `reference`

```
read_holding_register_string_list(reference: int, quantity: int, length: int) → List[bytes]
```

Read a list of strings from holding register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of bytes values read from `reference`

```
read_holding_register_ubint16(reference: int) → int
```

Read an unsigned 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint32(reference: int) → int
```

Read an unsigned 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint64(reference: int) → int
```

Read an unsigned 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ubint8(reference: int) → int
```

Read an unsigned 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ubint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint2143(reference: int) → int
```

Read an unsigned 2143-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint21436587(reference: int) → int
```

Read an unsigned 21436587-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 21436587-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 2143-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint3412(reference: int) → int
```

Read an unsigned 3412-endian 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:

int value read from `reference`

```
read_holding_register_uint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 3412-endian 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_uint78563412(reference: int) → int
```

Read an unsigned 78563412-endian 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_uint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 78563412-endian 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint16(reference: int) → int
```

Read an unsigned 16-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint32(reference: int) → int
```

Read an unsigned 32-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint64(reference: int) → int
```

Read an unsigned 64-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_holding_register_ulint8(reference: int) → int
```

Read an unsigned 8-bit integer from holding register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_holding_register_ulint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from holding register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register(reference: int) → int
```

Read a single input register from device memory.

Parameters:
reference – the register reference.
Returns:
the input register value

```
read_input_register_bdouble(reference: int) → float
```

Read a double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_bdouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:

list of float values read from `reference`

```
read_input_register_bfloat(reference: int) → float
```

Read a float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_bfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_bitfield(reference: int, length: int) → int
```

Read a bitfield from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
int value read from `reference`

```
read_input_register_bitfield_list(reference: int, quantity: int, length: int) →
List[int]
```

Read a list of bitfields from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct]
| Type[ByteType]) → ByteStruct | ByteType
```

Read data from input registers from device memory.

Parameters:

- **reference** – the register reference
- **data** – the type or instance or the decoder for the data

Returns:
instance of `datatype` containing the read data

```
read_input_register_double21436587(reference: int) → float
```

Read a 21436587-endian double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_double21436587_list(reference: int, quantity: int) → List[float]
```

Read a list of 21436587-endian doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_double78563412(reference: int) → float
```

Read a 78563412-endian double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_double78563412_list(reference: int, quantity: int) → List[float]
```

Read a list of 78563412-endian doubles from input register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_flag(reference: int) → bool
```

Read a bit from input register memory.

Parameters:
reference – the reference to read from
Returns:
bool value read from `reference`

```
read_input_register_flag_list(reference: int, quantity: int) → List[bool]
```

Read a list of bits from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of bool values read from `reference`

```
read_input_register_float2143(reference: int) → float
```

Read a 2143-endian float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_float2143_list(reference: int, quantity: int) → List[float]
```

Read a list of 2143-endian floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_float3412(reference: int) → float
```

Read a 3412-endian float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_float3412_list(reference: int, quantity: int) → List[float]
```

Read a list of 3412-endian floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_ldouble(reference: int) → float
```

Read a double from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_ldouble_list(reference: int, quantity: int) → List[float]
```

Read a list of doubles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_lfloat(reference: int) → float
```

Read a float from input register memory.

Parameters:
reference – the reference to read from
Returns:
float value read from `reference`

```
read_input_register_lfloat_list(reference: int, quantity: int) → List[float]
```

Read a list of floats from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of float values read from `reference`

```
read_input_register_nibble(reference: int) → int
```

Read a nibble from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_nibble_list(reference: int, quantity: int) → List[int]
```

Read a list of nibbles from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint16(reference: int) → int
```

Read a signed 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint32(reference: int) → int
```

Read a signed 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint64(reference: int) → int
```

Read a signed 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sbint8(reference: int) → int
```

Read a signed 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sbint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint2143(reference: int) → int
```

Read a signed 2143-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint21436587(reference: int) → int
```

Read a signed 21436587-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 21436587-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 2143-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint3412(reference: int) → int
```

Read a signed 3412-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 3412-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_sint78563412(reference: int) → int
```

Read a signed 78563412-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_sint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 78563412-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from

- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint16(reference: int) → int
```

Read a signed 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint16_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint32(reference: int) → int
```

Read a signed 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint32_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint64(reference: int) → int
```

Read a signed 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint64_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_slint8(reference: int) → int
```

Read a signed 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_slint8_list(reference: int, quantity: int) → List[int]
```

Read a list of signed 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_string(reference: int, length: int) → bytes
```

Read a string from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value

Returns:
bytes value read from `reference`

```
read_input_register_string_list(reference: int, quantity: int, length: int) →
List[bytes]
```

Read a list of strings from input register memory.

Parameters:

- **reference** – the reference to read from
- **length** – the length of the value
- **quantity** – the number of elements to read

Returns:
list of bytes values read from `reference`

```
read_input_register_ubint16(reference: int) → int
```

Read an unsigned 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint32(reference: int) → int
```

Read an unsigned 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint64(reference: int) → int
```

Read an unsigned 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ubint8(reference: int) → int
```

Read an unsigned 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ubint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint2143(reference: int) → int
```

Read an unsigned 2143-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint21436587(reference: int) → int
```

Read an unsigned 21436587-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint21436587_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 21436587-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint2143_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 2143-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint3412(reference: int) → int
```

Read an unsigned 3412-endian 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:

int value read from `reference`

```
read_input_register_uint3412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 3412-endian 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_uint78563412(reference: int) → int
```

Read an unsigned 78563412-endian 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_uint78563412_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 78563412-endian 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint16(reference: int) → int
```

Read an unsigned 16-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint16_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 16-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint32(reference: int) → int
```

Read an unsigned 32-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint32_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 32-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint64(reference: int) → int
```

Read an unsigned 64-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint64_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 64-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_input_register_ulint8(reference: int) → int
```

Read an unsigned 8-bit integer from input register memory.

Parameters:
reference – the reference to read from
Returns:
int value read from `reference`

```
read_input_register_ulint8_list(reference: int, quantity: int) → List[int]
```

Read a list of unsigned 8-bit integers from input register memory.

Parameters:

- **reference** – the reference to read from
- **quantity** – the number of elements to read

Returns:
list of int values read from `reference`

```
read_multiple_coils(reference: int, quantity: int) → List[int]
```

Read multiple coils from device memory.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of coils to be read.

Returns:
a list of coil states where each element is `1` if the corresponding coil is set and `0` else.

```
read_multiple_discrete_inputs(reference: int, quantity: int) → List[int]
```

Read multiple discrete inputs from device memory.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of discrete inputs to be read.

Returns:
a list of discrete inputs where each element is `1` if the corresponding discrete input is set and `0` else.

```
read_multiple_holding_registers(reference: int, quantity: int) → List[int]
```

Read multiple holding registers from device memory.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of registers to be read.

Returns:
the holding register values

```
read_multiple_input_registers(reference: int, quantity: int) → List[int]
```

Read multiple input registers from device memory.

Parameters:

- **reference** – the register reference.
- **quantity** – the number of registers to be read.

Returns:
the input register values

```
set_device_identifier(object_id: int | None, value: str | bytes | None) → None
```

Set a device identifier. The objects `"vendor_name"`, `"product_code"`, `"major_minor_version"`, `"vendor_url"`, `"product_name"`, `"model_name"` and `"user_application_name"` are predefined.

Set `value` to `None` to remove an entry.

Entries can be read using Modbus function code 43.14.

Parameters:

- **object_id** – the object identifier
- **value** – the value to be used

```
set_extended_memory(memory_number: int, records: Dict[int, List[int]] | None) → None
```

Set extended memory that can be read with Modbus function code 20 and written with Modbus function code 21.

If `records` is `None` the memory is removed.

Parameters:

- **memory_number** – the memory number (0x0001 to 0xFFFF)
- **records** – the records to be stored as a dict with key record numer and values list of integers

```
set_fifo_queue_content(address: int, content: List[int] | None) → None
```

Set the FIFO queue content.

Parameters:

- **address** – the address
- **content** – a list of integers or `None` to delete the queue contents

```
write_coil(reference: int, value: int) → None
```

Write a single coil to device memory.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written.

```
write_coil_bitfield(reference: int, value: int, length: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_coil_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_coil_data(reference: int, data: ByteStruct | ByteType | Type[ByteStruct] |
Type[ByteType]) → None
```

Write data to coils device memory.

Parameters:

- **reference** – the coil reference.
- **data** – the type or instance or the decoder for the data

```
write_coil_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_coil_flag_list(reference: int, values: List[bool]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_coil_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to coil memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_coil_nibble_list(reference: int, values: List[int]) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to coil memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_discrete_input(reference: int, value: int) → None
```

Read multiple discrete inputs from device memory

Parameters:

- **reference** – the register reference.
- **value** – the value to be written

Returns:

a list of discrete inputs where each element is `1` if the corresponding discrete input is set and `0` else.

```
write_discrete_input_bitfield(reference: int, value: int, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_discrete_input_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_discrete_input_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → None
```

Write data to discrete input device memory.

Parameters:

- **reference** – the discrete input reference.
- **data** – the type or instance or the decoder for the data

```
write_discrete_input_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_discrete_input_flag_list(reference: int, values: List[bool]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_discrete_input_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_discrete_input_nibble_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to discrete input memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register(reference: int, value: int) → None
```

Write single holding register to device memory.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written into the corresponding register.

```
write_holding_register_bdouble(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_bdouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_bfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_bfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_bitfield(reference: int, value: int, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_holding_register_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_holding_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → None
```

Write data to holding registers device memory.

Parameters:

- **reference** – the register reference.
- **data** – the type or instance or the decoder for the data

```
write_holding_register_double21436587(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 21436587-endian double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_double21436587_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 21436587-endian doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_double78563412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 78563412-endian double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_double78563412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 78563412-endian doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_holding_register_flag_list(reference: int, values: List[bool]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_holding_register_float2143(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 2143-endian float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_float2143_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 2143-endian floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_float3412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 3412-endian float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_float3412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 3412-endian floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_ldouble(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_ldouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_lfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_holding_register_lfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_holding_register_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_nibble_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sbint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sbint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 2143-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 21436587-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 21436587-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 2143-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 3412-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 3412-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_sint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 78563412-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_sint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 78563412-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_slint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_slint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_string(reference: int, value: bytes, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a string to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bytes value to be written
- **length** – the length of the value

```
write_holding_register_string_list(reference: int, values: List[bytes], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of strings to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bytes values to be written
- **length** – the length of the value

```
write_holding_register_ubint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ubint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ubint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 2143-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 21436587-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 21436587-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 2143-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 3412-endian 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 3412-endian 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_uint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 78563412-endian 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_uint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 78563412-endian 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_holding_register_ulint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to holding register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_holding_register_ulint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to holding register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register(reference: int, value: int) → None
```

Write single input register to device memory.

Parameters:

- **reference** – the register reference.
- **value** – the value to be written into the corresponding register.

```
write_input_register_bdouble(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_bdouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_bfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_bfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_bitfield(reference: int, value: int, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bitfield to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written
- **length** – the length of the value

```
write_input_register_bitfield_list(reference: int, values: List[int], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bitfields to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written
- **length** – the length of the value

```
write_input_register_data(reference: int, data: ByteStruct | ByteType |
Type[ByteStruct] | Type[ByteType]) → None
```

Write data to input registers device memory.

Parameters:

- **reference** – the register reference.
- **data** – the type or instance or the decoder for the data

```
write_input_register_double21436587(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 21436587-endian double to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_double21436587_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 21436587-endian doubles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_double78563412(reference: int, value: float) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 78563412-endian double to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_double78563412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 78563412-endian doubles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_flag(reference: int, value: bool) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a bit to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bool value to be written

```
write_input_register_flag_list(reference: int, values: List[bool]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of bits to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bool values to be written

```
write_input_register_float2143(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 2143-endian float to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_float2143_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 2143-endian floats to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_float3412(reference: int, value: float) → SerialRTUModbusResponse
| SerialASCIIModbusResponse | TCPModbusResponse
```

Write a 3412-endian float to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_float3412_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of 3412-endian floats to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_ldouble(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a double to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_ldouble_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of doubles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_lfloat(reference: int, value: float) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a float to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the float value to be written

```
write_input_register_lfloat_list(reference: int, values: List[float]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of floats to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of float values to be written

```
write_input_register_nibble(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a nibble to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_nibble_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of nibbles to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sbint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sbint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sbint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sbint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sbint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sbint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sbint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sbint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 2143-endian 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 21436587-endian 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 21436587-endian 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 2143-endian 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 3412-endian 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 3412-endian 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_sint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 78563412-endian 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_sint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 78563412-endian 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_slint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 16-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_slint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 16-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_slint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_slint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_slint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_slint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_slint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write a signed 8-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_slint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of signed 8-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_string(reference: int, value: bytes, length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a string to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the bytes value to be written
- **length** – the length of the value

```
write_input_register_string_list(reference: int, values: List[bytes], length: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of strings to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of bytes values to be written
- **length** – the length of the value

```
write_input_register_ubint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ubint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ubint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ubint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ubint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ubint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ubint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ubint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_uint2143(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 2143-endian 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_uint21436587(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 21436587-endian 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_uint21436587_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 21436587-endian 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_uint2143_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 2143-endian 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_uint3412(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 3412-endian 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_uint3412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 3412-endian 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_uint78563412(reference: int, value: int) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 78563412-endian 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_uint78563412_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 78563412-endian 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ulint16(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 16-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ulint16_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 16-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ulint32(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 32-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ulint32_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 32-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ulint64(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 64-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ulint64_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 64-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_input_register_ulint8(reference: int, value: int) → SerialRTUModbusResponse |
SerialASCIIModbusResponse | TCPModbusResponse
```

Write an unsigned 8-bit integer to input register memory.

Parameters:

- **reference** – the reference to write to
- **value** – the int value to be written

```
write_input_register_ulint8_list(reference: int, values: List[int]) →
SerialRTUModbusResponse | SerialASCIIModbusResponse | TCPModbusResponse
```

Write a list of unsigned 8-bit integers to input register memory.

Parameters:

- **reference** – the reference to write to
- **values** – a list of int values to be written

```
write_multiple_coils(reference: int, values: List[int]) → None
```

Write multiple coils to device memory.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

```
write_multiple_discrete_inputs(reference: int, values: List[int]) → None
```

Write multiple discrete inputs to device memory.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

```
write_multiple_holding_registers(reference: int, values: List[int]) → None
```

Write multiple holding registers to device memory.

Parameters:

- **reference** – the register reference.

- **values** – the values to be written.

```
write_multiple_input_registers(reference: int, values: List[int]) → None
```

Write multiple input registers to device memory.

Parameters:

- **reference** – the register reference.
- **values** – the values to be written.

Create Device Simulators with Special Behaviour

You can add special behaviour to a device to use it in tests.

For example, to simulate a device that raises an `illegal_data_address` when addresses greater than `1000` are accessed, run the following code:

```
from htf.modbus import ModbusSerialRTUDevice
from htf.modbus.types import ModbusRequest, ModbusResponse, RequestPDU

class TestDevice(ModbusSerialRTUDevice):
    def handle_request(self, adu: ModbusRequest) -> ModbusResponse:
        if adu.pdu.data.starting_address.get() > 1000:
            return self.create_exception_response(adu.pdu, exception_code="illegal_data_address")
        else:
            return super(TestDevice, self).handle_request(adu)
```

The following example disables the function code `3` (read multiple holding registers) completely, returning an `illegal function` exception:

```
from htf.modbus import ModbusSerialRTUDevice
from htf.modbus.types import ModbusRequest, ModbusResponse, RequestPDU

class TestDevice(ModbusSerialRTUDevice):
    def handle_function_code_3(self, pdu: RequestPDU) -> ModbusResponse:
        return self.create_exception_response(pdu, exception_code="illegal_function")
```

Exceptions

```
class htf.modbus.ModbusException
```

A Modbus exception raised by clients.

```
add_note(object, /)
```

Exception.add_note(note) – add a note to the exception

```
args
```

```
with_traceback(object, /)
```

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

```
class htf.modbus.ModbusNoResponseException
```

A Modbus exception raised by clients if no response was received.

```
add_note(object, /)
```

Exception.add_note(note) – add a note to the exception

```
args
```

```
with_traceback(object, /)
```

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

```
class htf.modbus.ModbusChecksumException(value: str, object: Any)
```

A Modbus exception raised if the checksum is broken.

```
add_note(object, /)
```

Exception.add_note(note) – add a note to the exception

```
args
```

```
with_traceback(object, /)
```

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

Memory

Memory is used to realize devices' memory.

Memory Base

Memory base-class.

```
class htf.modbus.MemoryBase(memory: MemoryBase | bytearray | None = None)
```

MemoryBase is the Modbus memory base-class.

Parameters:
memory – the memory to address or `None` for new bytearray

Bit Array

```
class htf.modbus.BitArray(memory: MemoryBase | bytearray | None = None)
```

BitArray is a bit-aligned Modbus memory for coils and discrete inputs.

Parameters:
memory – the memory to address or `None` for new bytearray

Register Array

```
class htf.modbus.RegisterArray(memory: MemoryBase | bytearray | None = None)
```

RegisterArray is a 2 byte-aligned Modbus memory for input and holding registers.

Parameters:
memory – the memory to address or `None` for new bytearray

Modbus Memory

```
class htf.modbus.ModbusMemory(shared_memory: bool = False)
```

ModbusMemory represents the physical memory of a Modbus device. It supports reading and writing of: coils, discrete inputs, input and holding registers.

Parameters:
shared_memory – If set to `True` a single shared memory is used.

```
read_coil(reference: int) → int
```

Read a coil.

Parameters:
reference – reference index of the coil
Returns:
value of the coil

```
read_coils_as_int(reference: int, quantity: int) → int
```

Read one or more coils as an int.

Parameters:

- **reference** – reference index of the first coil
- **quantity** – number of bits to read

Returns:
value of coils as integer

```
read_discrete_input(reference: int) → int
```

Read a discrete input bit.

Parameters:
reference – reference index of the discrete input
Returns:
value of the discrete input bit

```
read_discrete_input_as_int(reference: int, quantity: int) → int
```

Read one or more discrete inputs as an int.

Parameters:

- **reference** – reference index of the first discrete input
- **quantity** – number of bits to read

Returns:
value of discrete inputs as integer

```
read_holding_register(reference: int) → bytearray
```

Read a holding register.

Parameters:
reference – reference index of the holding register
Returns:
value of the holding register

```
read_input_register(reference: int) → bytearray
```

Read a input register.

Parameters:
reference – reference index of the input register
Returns:
value of the input register

```
write_coil(reference: int, value: int) → None
```

Write one coil.

Parameters:

- **reference** – reference index of the coil
- **value** – value to set the coil to

```
write_discrete_input(reference: int, value: int) → None
```

Write one discrete input.

Parameters:

- **reference** – reference index of the discrete input
- **value** – value to set the discrete input to

```
write_holding_register(reference: int, value: bytearray) → None
```

Write a holding register.

Parameters:

- **reference** – reference index of the holding register
- **value** – value to set the holding register to

```
write_input_register(reference: int, value: bytearray) → None
```

Write a input register.

Parameters:

- **reference** – reference index of the input register
- **value** – value to set the input register to

```
write_multiple_coils(reference: int, quantity: int, values: List[int]) → None
```

Write one or multiple coils.

Parameters:

- **reference** – reference index of the first coil
- **values** – values to set the coils to

```
write_multiple_discrete_inputs(reference: int, quantity: int, values: List[int]) → None
```

Write one or multiple discrete inputs.

Parameters:

- **reference** – reference index of the first discrete input
- **values** – values to set the discrete input to

```
write_multiple_holding_registers(reference: int, values: List[bytearray]) → None
```

Write one or multiple holding registers.

Parameters:

- **reference** – reference index of the first holding register
- **values** – values to set the holding register to

```
write_multiple_input_registers(reference: int, values: List[bytearray]) → None
```

Write one or multiple input registers.

Parameters:

- **reference** – reference index of the first input register
- **values** – values to set the input register to

Protocol

Requests and Responses

Requests and responses are used as Modbus communication messages for different transports.

Serial RTU Request

Serial RTU request.

```
class htf.modbus.SerialRTUModbusRequest
```

Modbus Serial RTU request.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Serial RTU Response

Serial RTU response.

```
class htf.modbus.SerialRTUModbusResponse
```

Modbus Serial RTU response.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.

Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Serial ASCII Request

Serial ASCII request.

```
class htf.modbus.SerialASCIIModbusRequest
```

Modbus Serial ASCII request.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Serial ASCII Response

Serial ASCII response.

```
class htf.modbus.SerialASCIIModbusResponse
```

Modbus Serial ASCII response.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.

- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

TCP Request

The TCP request object which is also used for TLS.

```
class htf.modbus.TCPModbusRequest
```

Modbus TCP request.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.

- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

TCP Response

The TCP response object which is also used for TLS.

```
class htf.modbus.TCPModbusResponse
```

Modbus TCP response.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.

- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Function Code

```
class htf.modbus.FunctionCode(value: int = 0, format: str | None = None)
```

Parameters:

- **value** – the initial value.
- **format=None** – Use “hex” to format values to hexadecimal. Use “bin” to format values to binary.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.

- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
get() → int
```

Return the value.

Returns:
the value.

```
get_byte_size() → int
```

Return the length of the byte type in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the length of the byte type in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
is_exception() → bool
```

Check if the current function code is an exception (msb is set).

Returns:
`True` if the current function code is an exception. Else `False`.

```
root() → ByteStruct | BitStruct
```

return root element

```
set(value: int) → None
```

Set the value.

Parameters:
value – the new value

```
set_fuzzing_values(values: Generator[Any, None, None] | List[Any] | None) → None
```

Set fuzzing values.

Parameters:
values – the values used for fuzzing.

```
up() → ByteStruct | BitStruct
```

Return the parent element.

PDU

```
class htf.modbus.ModbusPDUBase
```

ModbusPDUBase is the Modbus PDU base-class.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Request PDU

```
class htf.modbus.RequestPDU
```

RequestPDU is the request protocol data unit class.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.

- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
requests_generator() → Generator[Tuple[int, ByteStruct | BitStruct | ByteType |
BitType], None, None]
```

Yields function code, abstraction tuples.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Response PDU

```
class htf.modbus.ResponsePDU
```

ResponsePDU is the response protocol data unit class.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.

- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
responses_generator() → Generator[Tuple[int, ByteStruct | BitStruct | ByteType |
BitType], None, None]
```

Yields function code, abstraction tuples.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Exception Response

`htf.modbus.ExceptionResponse` is a base class for all Modbus responses. In case of an exception the payload is truncated automatically. Thus no special handling is needed.

```
class htf.modbus.ExceptionResponse
```

ExceptionResponse is the response class representing invalid requests or device failures. It's the base-class of all responses.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Read Coils Request

```
class htf.modbus.ReadCoilsRequest
```

Representation of a read coils request.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Read Coils Response

```
class htf.modbus.ReadCoilsResponse
```

Representation of a read coils response.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.

- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Read Discrete Inputs Request

```
class htf.modbus.ReadDiscreteInputsRequest
```

Representation of a read discrete inputs request.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.

Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.

Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Read Discrete Inputs Response

```
class htf.modbus.ReadDiscreteInputsResponse
```

Representation of a read discrete inputs response.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Read Holding Registers Request

```
class htf.modbus.ReadHoldingRegistersRequest
```

Representation of a read holding registers request.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Read Holding Registers Response

```
class htf.modbus.ReadHoldingRegistersResponse
```

Representation of a read holding registers response.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Read Input Registers Request

```
class htf.modbus.ReadInputRegistersRequest
```

Representation of a read input registers request.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Read Input Registers Response

```
class htf.modbus.ReadInputRegistersResponse
```

Representation of a read input registers response.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Write Single Coils Request

```
class htf.modbus.WriteSingleCoilRequest
```

Representation of a write single coil request.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Write Single Coils Response

```
class htf.modbus.WriteSingleCoilResponse
```

Representation of a write single coil response.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Write Single Register Request

```
class htf.modbus.WriteSingleRegisterRequest
```

Representation of a write single register request.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Write Single Register Response

```
class htf.modbus.WriteSingleRegisterResponse
```

Representation of a write single register response.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Read Exception Status Request

```
class htf.modbus.ReadExceptionStatusRequest
```

Representation of a read exception status request.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Read Exception Status Response

```
class htf.modbus.ReadExceptionStatusResponse
```

Representation of a read exception status response.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Variable Length Data

```
class htf.modbus.VariableLengthData(length: Callable | int | None = None, value: bytes |
str = b'', padding: bytes | str = b'\x00')
```

VariableLengthData is a class that holds an arbitrary amount of data. Its length depends on the protocol it's embedded in.

Parameters:

- **length=None** – states the string length. Can be a callable (e.g. lambda) or a scalar. If set to `None` the result is a null-terminated string. If set to an Integer the result is a fixed length string padded with `padding`. If set to a `callable` (lambda or function) the result is a fixed or variable length string padded with `padding`.
- **value=""** – the initial string value.
- **padding="x00"** – that padding pattern used for filling the encoded data if *value* is shorter than *length*.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
get() → bytes
```

Return the value.

```
get_byte_size() → int
```

Return the size in bytes.

```
get_size() → int
```

Return the size in bytes.

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
set(value: bytes | str) → None
```

Set the value.

Parameters:
value – the new value

```
set_fuzzing_values(values: Generator[Any, None, None] | List[Any] | None) → None
```

Set fuzzing values.

Parameters:
values – the values used for fuzzing.

```
set_length(length: Callable | int | None) → None
```

Set the length.

Parameters:
length=None – states the string length. Can be a callable (e.g. lambda) or a scalar. If set to `None` the result is a null-terminated string. If set to an Integer the result is a fixed length string padded with `padding`. If set to a `callable` (lambda or function) the result is a fixed or variable length string padded with `padding`.

```
up() → ByteStruct | BitStruct
```

Return the parent element.

Diagnostics Request

```
class htf.modbus.DiagnosticsRequest
```

Representation of a diagnostics request.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Diagnostics Response

```
class htf.modbus.DiagnosticsResponse
```

Representation of a diagnostics response.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Get Communication Event Counter Request

```
class htf.modbus.GetCommEventCounterRequest
```

Representation of a get communication event counter request.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Get Communication Event Counter Response

```
class htf.modbus.GetCommEventCounterResponse
```

Representation of a get communication event counter response.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Get Communication Event Log Request

```
class htf.modbus.GetCommEventLogRequest
```

Representation of a get communication event log request.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Get Communication Event Log Response

```
class htf.modbus.GetCommEventLogResponse
```

Representation of a get communication event log response.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Write Multiple Coils Request

```
class htf.modbus.WriteMultipleCoilsRequest
```

Representation of a write multiple coils request.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Write Multiple Coils Response

```
class htf.modbus.WriteMultipleCoilsResponse
```

Representation of a write multiple coils response.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Write Multiple Registers Request

```
class htf.modbus.WriteMultipleRegistersRequest
```

Representation of a write multiple registers request.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Write Multiple Registers Response

```
class htf.modbus.WriteMultipleRegistersResponse
```

Representation of a write multiple registers response.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Report Server ID Request

```
class htf.modbus.ReportServerIdRequest
```

Representation of a report server id request.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Report Server ID Response

```
class htf.modbus.ReportServerIdResponse
```

Representation of a report server id response.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Read Extended Memory Record Sub Request

```
class htf.modbus.ReadExtendedMemoryRecordSubRequest
```

Representation of a read extended memory record sub request.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Read Extended Memory Record Request

```
class htf.modbus.ReadExtendedMemoryRecordRequest
```

Representation of a read extended memory record request.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Read Extended Memory Record Response

```
class htf.modbus.ReadExtendedMemoryRecordResponse
```

Representation of a read extended memory record response.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Write Extended Memory Record Sub Request

```
class htf.modbus.WriteExtendedMemoryRecordSubRequest
```

Representation of a write extended memory record sub request.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Write Extended Memory Record Request

```
class htf.modbus.WriteExtendedMemoryRecordRequest
```

Representation of a write extended memory record request.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Write Extended Memory Record Response

```
class htf.modbus.WriteExtendedMemoryRecordResponse
```

Representation of a write extended memory record response.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Mask Write Request

```
class htf.modbus.MaskWriteRequest
```

Representation of a mask write request.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Mask Write Response

```
class htf.modbus.MaskWriteResponse
```

Representation of a mask write response.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Read Write Multiple Registers Request

```
class htf.modbus.ReadWriteMultipleRegistersRequest
```

Representation of a write multiple registers request.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Read Write Multiple Registers Response

```
class htf.modbus.ReadWriteMultipleRegistersResponse
```

Representation of a write multiple registers response.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Read FIFO Queue Request

```
class htf.modbus.ReadFifoQueueRequest
```

Representation of a read fifo queue request.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Read FIFO Queue Response

```
class htf.modbus.ReadFifoQueueResponse
```

Representation of a read fifo queue response.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Encapsulated Interface Transport Request

```
class htf.modbus.EncapsulatedInterfaceTransportRequest
```

Representation of an encapsulated interface transport request.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Encapsulated Interface Transport Response

```
class htf.modbus.EncapsulatedInterfaceTransportResponse
```

Representation of an encapsulated interface transport response.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Device Identification Object

```
class htf.modbus.ReadDeviceIdentificationObject
```

Representation of read device identification object.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

CRC

```
class htf.modbus.ModbusCrc
```

Calculates a reversed IBM-CRC-16 (0x8005) -> 0xa001 to be used with Modbus

```
crc(data: bytes) → int
```

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
get() → int
```

Return the value.

Returns:
the value.

```
get_byte_size() → int
```

Return the length of the byte type in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the length of the byte type in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
set(value: int) → None
```

Set the value.

Parameters:
value – the new value

```
set_fuzzing_values(values: Generator[Any, None, None] | List[Any] | None) → None
```

Set fuzzing values.

Parameters:
values – the values used for fuzzing.

```
up() → ByteStruct | BitStruct
```

Return the parent element.

LRC

```
class htf.modbus.ModbusLRC
```

Compute the logitudinal redudancy check for Modbus.

Parameters:

- **value** – the initial value.
- **format=None** – Use “hex” to format values to hexadecimal. Use “bin” to format values to binary.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
get() → int
```

Return the value.

Returns:
the value.

```
get_byte_size() → int
```

Return the length of the byte type in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the length of the byte type in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
lrc(data: bytes) → int
```

```
root() → ByteStruct | BitStruct
```

return root element

```
set(value: int) → None
```

Set the value.

Parameters:

value – the new value

```
set_fuzzing_values(values: Generator[Any, None, None] | List[Any] | None) → None
```

Set fuzzing values.

Parameters:
values – the values used for fuzzing.

```
up() → ByteStruct | BitStruct
```

Return the parent element.

Changelog

htf-modbus-3.2.0

- ship `cp312-abi3` wheels that are compatible with Python 3.12 and newer versions

htf-modbus-3.1.11

- implement Cyber Resilience Act Conformity (CRA)
- update dependencies

htf-modbus-3.1.10

- fix extended memory implementation in `htf.modbus.ModbusTCPDevice` and `htf.modbus.ModbusUDPDevice`

htf-modbus-3.1.9

- add serial parameters to device simulators
- add methods to break checksums in clients, servers and devices for testing purposes
- fix bit order for function code 15
- fix LRC calculation

htf-modbus-3.1.8

- add support for Python 3.13

htf-modbus-3.1.7

- fix handling of address `0xFFFF` in `htf.modbus.ModbusTCPDevice` and `htf.modbus.ModbusUDPDevice`

htf-modbus-3.1.6

- add requirements for validation purposes

htf-modbus-3.1.5

- add build environment data for validation purposes

htf-modbus-3.1.4

- `htf-modbus` can now be used standalone

htf-modbus-3.1.3

- add support for Python 3.12

htf-modbus-3.1.2

- improve connection handling in `htf.modbus.ModbusTCPDevice` and `htf.modbus.ModbusUDPDevice`

htf-modbus-3.1.1

- fix not correctly closed TCP connections in `htf.modbus.ModbusTCPDevice`
- add timeout to `htf.modbus.ModbusTCPDevice`

htf-modbus-3.1.0

- add type-hints
- use `hlm-3.1`
- use `oser-3.1`

htf-modbus-3.0.2

- add support for oser-3.1

htf-modbus-3.0.1

- add support for Python 3.11

htf-modbus-3.0.0

- extract `htf-modbus` from `htf`

HART — The HART Testing Framework

The HART Testing Framework enables users to

- write automatable tests for HART slaves
- simulate HART devices and HART masters
- add device specific commands
- develop and debug a HART device interactively using the HART-Shell
- monitor HART messages using the HART-Monitor

- send corrupted frames and
- run HART Fuzzing tests

It is based on version 7.6 of the HART specification and implements all universal and common practice commands (0..533). Extended commands up to 16-bit are supported.

You need the `hart` feature in your license to unlock the HART Testing Framework.

HART Interface

The `HartInterface` implements the data link layer and physical layer. It enables you to write binary data that is sent to the connected HART devices and read HART frames.

The used serial port must support the `RS485 mode` that sets `RTS` high while sending. Many USB-Uarts do not support this feature and thus cannot be used.

It is recommended to use USB-HART-Modems such as the MACTek Viator. They do support this feature and are known to work.

Real COM ports are very sparse nowadays.

```
class htf.hart.HartInterface(comport: str, timeout: int | float = 0.3)
```

Data-Link-Layer and Physical-Layer for HART.

Parameters:

- **comport** – the comport to be used where the HART modem is connected to. Must support RS485 mode.
- **timeout=.3** – the time out for a single byte. Timeouts in `read()` are multiple of this timeout.

```
close() → None
```

Close the HART interface. Is called in __del__, too.

```
get_last_preamble_count() → int
```

Gets the count of preambles that was most recently received. This does **NOT** require a full HART frame to be read, but does require the preambles to be followed by a valid delimiter.

Returns:
the most recently received number of consecutive preambles
Return type:
int

```
query(data: bytes, number_of_preambles: int = 5, preamble: bytes = b'\xff', timeout: int
| float = 1.0) → bytes | None
```

Query a HART command.

Parameters:

- **request** – the hart frame to be sent.
- **number_of_preambles=5** – the number of preambles.
- **preamble='\xff'** – `number_of_preambles` times `preamble` is sent before the frame content is sent.
- **timeout=1.0** – the timeout in seconds.

Returns:
if no frame was received `None` is returned else a HART frame is returned.
Return type:
bytes or None

```
read(timeout: int | float = 1.0) → bytes | None
```

Read a frame.

Parameters:
timeout=1.0 – the timeout in seconds.
Returns:
if no frame was received `None` is returned else a HART frame is returned.
Return type:
bytes or None

```
write(data: bytes, number_of_preambles: int = 5, preamble: bytes = b'\xff') → None
```

Write a HART frame.

Parameters:

- **data** – the bytes to be sent.
- **number_of_preambles=5** – the number of preambles.
- **preamble='\xff'** – `number_of_preambles` times `preamble` is sent before the frame content is sent.

Example:

```
import oser
from htf.hart import HartInterface

hi = HartInterface(comport="/dev/ttyUSB1")
cmd0_request = b"\x02\x80\x00\x00\x82"
response = hi.query(cmd0_request)
print(oser.to_hex(response))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18
\x06\x80\x00\x0e\x00\x00\x00\x13\x37\x00\x00\x00\x00\x00\x00\x00\x00\x23\x8f
hi.close()
```

HART Frame

The `HartFrame` abstracts the HART-PDU that is used for HART communication.

It is based on OSER and lets the user specify device specific commands.

The check byte is generated automatically when `encode` is called.

```
class htf.hart.HartFrame(command: int | None = None, auto_length: bool = True,
auto_checkbyte: bool = True)
```

A HART frame abstraction.

Parameters:

- **command=None** – the HART command number.
- **auto_length=True** – if set to `True` the byte count of the HART frame is calculated automatically when calling `encode`.
- **auto_checkbyte=True** – if set to `True` the checkbyte of the HART frame is calculated automatically when calling `encode`.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string. `byte_count` is set automatically if `auto_length` is supplied.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_command_request_parser(command: int) → ByteStruct
```

Get a command request parser.

Parameters:
command – the command number to be returned.
Returns:
the command parser.
Return type:
ByteStruct
Raises:

- **KeyError** – Raises a `KeyError` if the command if the command
- **is not set.** –

```
get_command_response_parser(command: int) → ByteStruct
```

Get a command response parser.

Parameters:
command – the command number to be returned.
Returns:

the command parser.
Return type:
ByteStruct
Raises:

- **KeyError** – Raises a `KeyError` if the command if the command
- **is not set.** –

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
requests_generator() → Generator[Tuple[int, ByteStruct | BitStruct | ByteType |
BitType], None, None]
```

A generator that yields command, parser tuples for requests.

This method shall be overwritten to realize device specific commands.

Yields:
tuples of command and parser

```
responses_generator() → Generator[Tuple[int, ByteStruct | BitStruct | ByteType |
BitType], None, None]
```

A generator that yields command, parser tuples for responses.

This method shall be overwritten to realize device specific commands.

Yields:
tuples of command and parser

```
root() → ByteStruct | BitStruct
```

return root element

```
set_auto_check_byte_enable(enable_auto_check_byte: bool = True) → None
```

Enable or disable the automatic checkbyte calculation on `encode`.

Parameters:
enable_auto_check_byte=True – the new enable state.

```
set_auto_length_enable(enable_auto_length: bool = True) → None
```

Enable or disable the auto length feature.

Parameters:
enable_auto_length=True – the new enable state.

```
set_command_request_parser(command: int, parser: ByteStruct | BitStruct | ByteType |
BitType) → None
```

Set a command request parser. This overwrites exising command parsers.

Parameters:

- **command** – the command number to be set.
- **parser** – the parser for the command.

```
set_command_response_parser(command: int, parser: ByteStruct | BitStruct | ByteType |
BitType) → None
```

Set a command response parser. This overwrites exising command parsers.

Parameters:

- **command** – the command number to be set.
- **parser** – the parser for the command.

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Example:

```
import oser
from htf.hart import HartFrame

hf = HartFrame(1)
hf.address.address.set(0x13370023)
encoded = hf.encode()
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'stx' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: 0 (Flag)
        address: 322371619 (BitField(38))
    expansion_bytes: Array():
    [
    ]
    command: 1 (UBInt8)
    byte_count: 0 (UBInt8)
    payload: Command1_ReadPrimaryVariableRequest():
    check_byte: 4 (CheckByte)

print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x13\x37\x00\x23\x01\x00\x04
```

HART Application Layer

The HART Application Layer interfaces the `HartInterface` and the abstracted protocol with `HartFrame`.

It lets the user communicate with a single HART device easily.

Additionally you can search for HART devices, set the master address and the device address.

```
class htf.hart.HartApplicationLayer(interface: HartInterface, decoder_generator:
Type[HartFrame] | Callable[[...], HartFrame] | None = None)
```

HART Application-Layer.

Parameters:

- **interface** – the HART interface to be used.
- **decoder_generator=None** – a callable that returns a HartFrame instance with or without device specific commands.

```
close() → None
```

Close the HART application layer and close the HART interface supplied. Is called in __del__, too.

```
find_device(polling_addresses: List[int] | None = None) → int
```

Find the first HART device by polling all addresses stated in `polling_addresses` and use the first device found.

If no device is found an `AssertionError` is raised.

Parameters:
polling_addresses=[0] – the polling addresses used to find a HART device.
Returns:
Unique HART address of the first device found in polling.
Return type:
int

```
poll_devices(polling_addresses: List[int] | None = None) → Generator[int, None, None]
```

Poll `polling_addresses` to find HART devices.

Parameters:
polling_addresses=[0] – list of HART polling addresses
Yields:
int – HART device addresses

```
query(request: HartFrame, number_of_preambles: int = 5, preamble: bytes = b'\xff',
timeout: int | float = 1.0) → HartFrame | None
```

Query a HART command. The HART frame is encoded automatically.

Parameters:

- **request** – the hart frame to be sent.
- **number_of_preambles=5** – the number of preambles.
- **preamble='ÿ'** – `number_of_preambles` times `preamble` is sent before the frame content is sent.
- **timeout=1.0** – the timeout in seconds.

Returns:
if no frame was received `None` is returned.
Return type:
HartFrame or None

```
read(timeout: int | float = 1.0) → HartFrame | None
```

Read a message using the interface and decode it using the decoder.

Parameters:

timeout=1.0 – the timeout in seconds.
Returns:
if no frame was received `None` is returned.
Return type:
HartFrame or None

```
set_decoder_generator(decoder_generator: Type[HartFrame] | Callable[[...], HartFrame] |
None = None) → None
```

Set the frame generator.

Parameters:
decoder_generator=HartFrame – when called returns

```
set_device_address(device_address: int) → None
```

Set the current HART device address.

Parameters:
device_address – the HART device address to be used (<= 38 bits)

```
set_master_address(master_address: str) → None
```

Set the current simulated master address.

Parameters:
master_address – “primary_master” or “secondary_master”

```
write(frame: HartFrame, number_of_preambles: int = 5, preamble: bytes = b'\xff') → None
```

Write a HART frame. The HART frame is encoded automatically.

Except for command 0 the HART device address is set automatically before sending the frame content.

Parameters:

- **frame** – the hart frame to be sent.
- **number_of_preambles=5** – the number of preambles.
- **preamble='ÿ'** – `number_of_preambles` times `preamble` is sent before the frame content is sent.

Find device at polling address 0 manually:

```
import oser
from htf.hart import \
     HartInterface, HartApplicationLayer, HartFrame

hi = HartInterface(comport="/dev/ttyUSB1")
```

```
hal = HartApplicationLayer(hi)

# search for HART device at polling address 0 manually
cmd0_request = HartFrame(0)
cmd0_request.delimiter.frame_type.set("stx")
cmd0_request.delimiter.address_type.set("polling")
cmd0_request.address.address.set(0)
print(cmd0_request)
HartFrame():
    delimiter: Delimiter():
        address_type: 'polling' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'stx' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: 0 (Flag)
        address: 0 (BitField(6))
    expansion_bytes: Array():
    [
    ]
    command: 0 (UBInt8)
    byte_count: 0 (UBInt8)
    payload: Command0_ReadUniqueIdentifierRequest():
    check_byte: 0 (CheckByte)

response = hal.query(cmd0_request)
print(response)
HartFrame():
    delimiter: Delimiter():
        address_type: 'polling' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: 0 (Flag)
        address: 0 (BitField(6))
    expansion_bytes: Array():
    [
    ]
    command: 0 (UBInt8)
    byte_count: 14 (UBInt8)
    payload: Command0_ReadUniqueIdentifierResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        data_object_id: 0 (UBInt8)
        device_type: 4919 (UBInt16)
        minimum_preambles_request: 0 (UBInt8)
        protocol_revision: 0 (UBInt8)
        device_revision_level: 0 (UBInt8)
        software_revision_level: 0 (UBInt8)
        hardware_revision_level_and_physical_signaling_code: 0 (UBInt8)
        flags: 0 (UBInt8)
        device_id: BitStruct():
            value: 35 (BitField(24))
    check_byte: 143 (CheckByte)

device_address = (response.payload.device_type.get() << 24) +\
     response.payload.device_id.value.get()
```

```
print("0x%x" % device_address)
0x1337000023
hal.close()
```

Find device at polling address 0:

```
import oser
from htf.hart import \
     HartInterface, HartApplicationLayer

hi = HartInterface(comport="/dev/ttyUSB1")
hal = HartApplicationLayer(hi)

# search for HART device at polling address 0
device_address = hal.find_device()
print("0x%x" % device_address)
0x1337000023
hal.close()
```

Poll addresses 0 .. 9:

```
import oser
from htf.hart import \
     HartInterface, HartApplicationLayer

hi = HartInterface(comport="/dev/ttyUSB1")
hal = HartApplicationLayer(hi)

# poll address 0 .. 9
device_addresses = hal.poll_devices(range(10))
for device_address in device_addresses:
    print("0x%x" % device_address)

0x1337000023

hal.close()
```

Query Command 1 from first found HART device and print primary variable:

```
import oser
from htf.hart import \
     HartInterface, HartApplicationLayer, HartFrame

hi = HartInterface(comport="/dev/ttyUSB1")
hal = HartApplicationLayer(hi)

# search for HART device at polling address 0
hal.find_device()
82527125539

# query command 1
cmd1_request = HartFrame(1)
```

```
response = hal.query(cmd1_request)

print(response)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: 0 (Flag)
        address: 82527125539 (BitField(38))
    expansion_bytes: Array():
    [
    ]
    command: 1 (UBInt8)
    byte_count: 7 (UBInt8)
    payload: Command1_ReadPrimaryVariableResponse():
        response_code: 'buffer_overflow' (UBInt8)
        field_device_status: non-primary_variable_out_of_limits (FieldDeviceStatus)
        units: 123 (UBInt8)
        primary_variable: 456.78900146484375 (BFloat)
    check_byte: 193 (CheckByte)

primary_variable = response.payload.primary_variable.get()
print("PV:", primary_variable)
PV: 456.78900146484375
hal.close()
```

Data Types

The data types implement the needed components to build HART frames.

Packed ASCII

`PackedASCII` implements the `Packed ASCII` data type for the HART protocol.

```
class htf.hart.data_types.PackedASCII(length: Callable | int | None = None, value: bytes |
str = b'', padding: bytes | str = b'\x00')
```

The HART Packed ASCII data type

Parameters:

- **length=None** – states the string length. Can be a callable (e.g. lambda) or a scalar. If set to an Integer the result is a fixed length string padded with `padding`. If set to a `callable` (lambda or function) the result is a fixed or variable length string padded with `padding`. The length must be a multiple of three.
- **value=""** – the initial string value.
- **padding="\x00"** – that padding pattern used for filling the encoded data if *value* is shorter than *length*.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the encoded binary string.
Return type:
bytes

```
get() → bytes
```

Return the value.

```
get_byte_size() → int
```

Return the size in bytes.

```
get_size() → int
```

Return the size in bytes.

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
set(value: bytes | str) → None
```

Set the value.

Parameters:
value – the new value

```
set_fuzzing_values(values: Generator[Any, None, None] | List[Any] | None) → None
```

Set fuzzing values.

Parameters:
values – the values used for fuzzing.

```
set_length(length: Callable | int | None) → None
```

Set the length.

Parameters:
length=None – states the string length. Can be a callable (e.g. lambda) or a scalar. If set to `None` the result is a null-terminated string. If set to an Integer the result is a fixed length string padded with `padding`. If set to a `callable` (lambda or function) the result is a fixed or variable length string padded with `padding`.

```
up() → ByteStruct | BitStruct
```

Return the parent element.

Example:

```
import oser
from htf.hart import PackedASCII

pa = PackedASCII(length=6, value=b"ABCDEFGH")
encoded = pa.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5
\x04\x20\xc4\x14\x61\xc8
pa2 = PackedASCII(length=3, value=b"ABC")
```

```
pa2.decode(encoded)
3
print(pa2)
b'ABCD'
```

Date

`Date` implements the `Date` data type for the HART protocol.

```
class htf.hart.data_types.Date(date: date)
```

The HART Date data type

Parameters:
date – an instance of `datetime.date` or `datetime.datetime` between `1900-01-01` and `2155-12-31`.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the encoded binary string.
Return type:
bytes

```
get() → date
```

Return the value.

```
get_byte_size() → int
```

Return the size in bytes.

```
get_size() → int
```

Return the size in bytes.

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
set(value: date) → None
```

Set the value.

Parameters:
value – the new value

```
set_fuzzing_values(values: Generator[Any, None, None] | List[Any] | None) → None
```

Set fuzzing values.

Parameters:
values – the values used for fuzzing.

```
set_length(length: Callable | int | None) → None
```

Set the length.

Parameters:

length=None – states the string length. Can be a callable (e.g. lambda) or a scalar. If set to `None` the result is a null-terminated string. If set to an Integer the result is a fixed length string padded with `padding`. If set to a `callable` (lambda or function) the result is a fixed or variable length string padded with `padding`.

```
up() → ByteStruct | BitStruct
```

Return the parent element.

Example:

```
import oser
from htf.hart import Date
import datetime

date = Date(datetime.date(2017, 1, 11))
encoded = date.encode()
print(oser.to_hex(encoded))
   0|  1|  2
\x75\x01\x0b
date2 = Date(None)
date2.decode(encoded)
3
print(date2)
datetime.date(2017, 1, 11)
```

Time

`Time` implements the `Time` data type for the HART protocol.

```
class htf.hart.data_types.Time(time: time)
```

The HART Time data type.

Parameters:
time – an instance of `datetime.time` or `datetime.datetime` between `00:00:00` and `23:59:59`.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.

Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
get() → time
```

Return the value.

Returns:
the value.

```
get_byte_size() → int
```

Return the length of the byte type in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the length of the byte type in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
set(value)
```

Set the value.

Parameters:
value – the new value

```
set_fuzzing_values(values: Generator[Any, None, None] | List[Any] | None) → None
```

Set fuzzing values.

Parameters:
values – the values used for fuzzing.

```
up() → ByteStruct | BitStruct
```

Return the parent element.

Example:

```
import oser
from htf.hart import Time

import datetime

time = Time(datetime.time(23, 24, 7, 1000))
encoded = time.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3
\xa0\xb0\x3b\x20
time2 = Time(None)
time2.decode(encoded)
4
print(time2)
23:24:07.001000 (Time)
```

Response Code

`ResponseCode` implements the `Response Code` data type for the HART protocol.

When printed the states are concatenated with `;`.

```
class htf.hart.data_types.ResponseCode(**kwargs: int)
```

HART device specific response code generator.

`0: "success"` is added automatically.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** (*bytes*) – the data buffer that is decoded.
- **full_data** (*bytes*) – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** (*bytes*) – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** (*bytes*) – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** (*bytes*) – the binary data of the current context. The user normally does not need to supply this.

Returns:
the encoded binary string.
Return type:
bytes

```
get() → str
```

Return the value.

Returns:
the value

```
get_byte_size() → int
```

Return the size in bytes.

```
get_size() → int
```

Return the size in bits or bytes, depending on the prototype.

```
get_value() → Any
```

Return the raw value.

Returns:
the raw value.
Return type:
object

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None (*object*) – stop introspection at `stop_at`.

```
is_error() → bool
```

Checks if the response code is of class ‘error’ or not. Warning response codes must have the ‘warning_’-prefix. If there is no prefix the response is considered to be an error.

```
root() → ByteStruct | BitStruct
```

return root element

```
set(value: Any) → None
```

Set the value.

Parameters:
value – the new value

```
set_fuzzing_values(values: Generator[Any, None, None] | List[Any] | None) → None
```

Set fuzzing values.

Parameters:

values (*iterable*) – the values used for fuzzing.

```
up() → ByteStruct | BitStruct
```

Return the parent element.

If the MSB of the response code's value is set a communication error is represented:

```
import oser
from htf.hart import ResponseCode

response_code = ResponseCode()
response_code.set(0x82)
encoded = response_code.encode()
print(oser.to_hex(encoded))
   0
\x82
print(response_code)
'buffer_overflow' (UBInt8)


response_code2 = ResponseCode()
response_code2.decode(b"\xff")
1
print(response_code2)
'vertical_perity_error; overrun_error; framing_error; longitudinal_error; buffer_overflow' (UBInt8)
```

If the MSB is not set a command specific response code can be represented. Command specific response codes are passed to `ResponseCode` as named parameters.

```
import oser
from htf.hart import ResponseCode


class Command48_ReadAdditionalDeviceStatusResponse(oser.ByteStruct):
     """
     HART command 48 response abstraction.
     """
     def __init__(self):
         super(Command48_ReadAdditionalDeviceStatusResponse, self).__init__()

         self.response_code = ResponseCode(
                 error_device_specific_command_error=6,
                 warning_update_in_progress=8,
                 error_access_restricted=16,
                 error_busy=32)
         self.data = oser.String(length=3)
         # ...

c48 = Command48_ReadAdditionalDeviceStatusResponse()
c48.response_code.set(32)
print(c48)
Command48_ReadAdditionalDeviceStatusResponse():
```

```
    response_code: 'error_busy' (UBInt8)
    data: b''


encoded = c48.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3
\x20\x00\x00\x00

c48_2 = Command48_ReadAdditionalDeviceStatusResponse()
c48_2.decode(b"\x08\x00\x00\x00")
4
print(c48_2)
Command48_ReadAdditionalDeviceStatusResponse():
    response_code: 'warning_update_in_progress' (UBInt8)
    data: b'\x00\x00\x00'
```

Field Device Status

`FieldDeviceStatus` implements the `Field Device Status` data type for the HART protocol.

When printed the states are concatenated with `;`.

```
class htf.hart.data_types.FieldDeviceStatus(value: int = 0, format: str | None = None)
```

HART Field Device Status data type.

Parameters:

- **value** – the initial value.
- **format=None** – Use “hex” to format values to hexadecimal. Use “bin” to format values to binary.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
get() → int
```

Return the value.

Returns:
the value.

```
get_byte_size() → int
```

Return the length of the byte type in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the length of the byte type in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
set(value: int) → None
```

Set the value.

Parameters:
value – the new value

```
set_fuzzing_values(values: Generator[Any, None, None] | List[Any] | None) → None
```

Set fuzzing values.

Parameters:
values – the values used for fuzzing.

```
up() → ByteStruct | BitStruct
```

Return the parent element.

The `FieldDeviceStatus` is a bit field for predefined values:

```
import oser
from htf.hart import FieldDeviceStatus

fds = FieldDeviceStatus(0x42)
print(fds)
configuration_changed; non-primary_variable_out_of_limits (FieldDeviceStatus)


fds2 = FieldDeviceStatus(0x02)
print(fds2)
non-primary_variable_out_of_limits (FieldDeviceStatus)


encoded = fds2.encode()
print(oser.to_hex(encoded))
   0
\x02

fds3 = FieldDeviceStatus(0x00)
fds3.decode(encoded)
1
print(fds3)
non-primary_variable_out_of_limits (FieldDeviceStatus)
```

HART Responses

`HartResponse` implements a `HART response` data type to automatically handle error responses. All HART responses should be derived from this class.

```
class htf.hart.data_types.HartResponse
```

A ByteStruct for HART responses which checks for error response codes in _items() and stops iteration after field_device_status. All HART responses should be derived from this class.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string and return the number of bytes that were decoded.

Parameters:

- **data** – the data buffer that is decoded.
- **full_data** – the binary data string until the part to be decoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

Returns:
the number of bytes that were decoded.
Return type:
int

```
encode(full_data: bytes = b'', context_data: bytes = b'') → bytes
```

Return the encoded binary string.

Parameters:

- **full_data** – the binary data string until the part to be encoded. The user normally does not need to supply this.
- **context_data** – the binary data of the current context. The user normally does not need to supply this.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data – data to be used to fill the calling instance.

```
fuzzing_iterator(copy: bool = False) → Generator[Any, None, None]
```

The fuzzing iterator iterates over all combinations of set fuzzing values (`oser.ByteType.set_fuzzing_values`). If no fuzzing values are set the current struct is yielded.

Parameters:
copy=False – if set to `True` the generated fuzzing values are deep copies of the original. Creating these deep copies is slow. If set to `False` the original struct is retruned and the generated value must be used immediately since the next generated overwrites the values on the same value.
Yields:
the fuzzing combinations.

```
get_byte_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
get_size() → int
```

Return the size in bytes.

Returns:
the length of the byte type in bytes.
Return type:
int

```
introspect(stop_at: ByteStruct | BitStruct | ByteType | BitType | None = None) → str
```

Return the introspection representation of the object as a string.

Parameters:
stop_at=None – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Universal Commands

Command 0

Command 0 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(0)
print(hf)
    HartFrame():
```

```
        delimiter: Delimiter():
            address_type: 'polling' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (PollingAddress(6))
        expansion_bytes: Array():
        [
        ]
        command: 0 (Command)
        byte_count: 0 (UBInt8)
        payload: Command0_ReadUniqueIdentifierRequest():
        check_byte: 130 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4
\x02\x80\x00\x00\x82
# response
hf = HartFrame(0)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'polling' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (PollingAddress(6))
    expansion_bytes: Array():
    [
    ]
    command: 0 (Command)
    byte_count: 14 (UBInt8)
    payload: Command0_ReadUniqueIdentifierResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        data_object_id: 0 (UBInt8)
        device_type: 0 (UBInt16)
        minimum_preambles_request: 0 (UBInt8)
        protocol_revision: 0 (UBInt8)
        device_revision_level: 0 (UBInt8)
        software_revision_level: 0 (UBInt8)
        hardware_revision_level_and_physical_signaling_code: 0 (UBInt8)
        flags: 0 (UBInt8)
        device_id: BitStruct():
            value: 0 (BitField(24))
    check_byte: 136 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18
\x06\x80\x00\x0e\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x88
```

Command 1

Command 1 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(1)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 1 (Command)
        byte_count: 0 (UBInt8)
        payload: Command1_ReadPrimaryVariableRequest():
        check_byte: 3 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x01\x00\x03
# response
hf = HartFrame(1)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 1 (Command)
    byte_count: 7 (UBInt8)
    payload: Command1_ReadPrimaryVariableResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        units: 0 (UBInt8)
        primary_variable: 0.0 (BFloat)
    check_byte: 0 (CheckByte)

encoded = hf.encode()
```

```
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15
\x86\x80\x00\x00\x00\x00\x01\x07\x00\x00\x00\x00\x00\x00\x00\x00
```

Command 2

Command 2 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(2)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 2 (Command)
        byte_count: 0 (UBInt8)
        payload: Command2_ReadLoopCurrentAndPercentageOfRangeRequest():
        check_byte: 0 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x02\x00\x00
# response
hf = HartFrame(2)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 2 (Command)
    byte_count: 10 (UBInt8)
    payload: Command2_ReadLoopCurrentAndPercentageOfRangeResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
```

```
        primary_variable_loop_current: 0.0 (BFloat)
        primary_variable_percentage_of_range: 0.0 (BFloat)
    check_byte: 14 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18
\x86\x80\x00\x00\x00\x00\x02\x0a\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0e
```

Command 3

Command 3 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(3)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 3 (Command)
        byte_count: 0 (UBInt8)
        payload: Command3_ReadDynamicVariablesAndLoopCurrentRequest():
        check_byte: 1 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x03\x00\x01
# response
hf = HartFrame(3)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
```

```
    command: 3 (Command)
    byte_count: 6 (UBInt8)
    payload: Command3_ReadDynamicVariablesAndLoopCurrentResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        primary_variable_loop_current: 0.0 (BFloat)
        units_and_variables: Array():
        [
        ]
    check_byte: 3 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14
\x86\x80\x00\x00\x00\x00\x03\x06\x00\x00\x00\x00\x00\x00\x03
```

Command 6

Command 6 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(6)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 6 (Command)
        byte_count: 1 (UBInt8)
        payload: Command6_WritePollingAddressRequest():
            polling_address: 0 (UBInt8)
        check_byte: 5 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x82\x80\x00\x00\x00\x00\x06\x01\x00\x05
# response
hf = HartFrame(6)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
```

```
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 6 (Command)
    byte_count: 4 (UBInt8)
    payload: Command6_WritePollingAddressResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        polling_address: 0 (UBInt8)
        loop_current_mode: 0 (UBInt8)
    check_byte: 4 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12
\x86\x80\x00\x00\x00\x00\x06\x04\x00\x00\x00\x00\x04
```

Command 7

Command 7 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(7)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 7 (Command)
        byte_count: 0 (UBInt8)
        payload: Command7_ReadLoopConfigurationRequest():
        check_byte: 5 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x07\x00\x05
# response
hf = HartFrame(7)
hf.delimiter.frame_type.set("ack")
print(hf)
```

```
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 7 (Command)
    byte_count: 4 (UBInt8)
    payload: Command7_ReadLoopConfigurationResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        polling_address: 0 (UBInt8)
        loop_current_mode: 0 (UBInt8)
    check_byte: 5 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12
\x86\x80\x00\x00\x00\x00\x07\x04\x00\x00\x00\x00\x05
```

Command 8

Command 8 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(8)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 8 (Command)
        byte_count: 0 (UBInt8)
        payload: Command8_ReadDynamicVariableClassificationsRequest():
        check_byte: 10 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
```

```
\x82\x80\x00\x00\x00\x00\x08\x00\x0a
# response
hf = HartFrame(8)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 8 (Command)
    byte_count: 6 (UBInt8)
    payload: Command8_ReadDynamicVariableClassificationsResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        primary_variable_classification: 0 (UBInt8)
        secondary_variable_classification: 0 (UBInt8)
        tertiary_variable_classification: 0 (UBInt8)
        quaternary_variable_classification: 0 (UBInt8)
    check_byte: 8 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14
\x86\x80\x00\x00\x00\x00\x08\x06\x00\x00\x00\x00\x00\x00\x08
```

Command 9

Command 9 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(9)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 9 (Command)
```

```
        byte_count: 0 (UBInt8)
        payload: Command9_ReadDeviceVariablesWithStatusRequest():
            device_variable_codes: Array():
            [
            ]
        check_byte: 11 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x09\x00\x0b
# response
hf = HartFrame(9)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 9 (Command)
    byte_count: 7 (UBInt8)
    payload: Command9_ReadDeviceVariablesWithStatusResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        extended_field_device_status: 0 (UBInt8)
        variables_and_statuses: Array():
        [
        ]
        slot0_data_time_stamp: 00:00:00 (Time)
    check_byte: 8 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15
\x86\x80\x00\x00\x00\x00\x09\x07\x00\x00\x00\x00\x00\x00\x00\x08
```

Command 11

Command 11 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(11)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
```

```
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 11 (Command)
        byte_count: 6 (UBInt8)
        payload: Command11_ReadUniqueIdentifierAssociatedWithTagRequest():
            tag: b'????????'
        check_byte: 15 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14
\x82\x80\x00\x00\x00\x00\x0b\x06\xff\xff\xff\xff\xff\xff\x0f
# response
hf = HartFrame(11)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 11 (Command)
    byte_count: 14 (UBInt8)
    payload: Command11_ReadUniqueIdentifierAssociatedWithTagResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        data_object_id: 0 (UBInt8)
        device_type: 0 (UBInt16)
        minimum_preambles_request: 0 (UBInt8)
        protocol_revision: 0 (UBInt8)
        device_revision_level: 0 (UBInt8)
        software_revision_level: 0 (UBInt8)
        hardware_revision_level_and_physical_signaling_code: 0 (UBInt8)
        flags: 0 (UBInt8)
        device_id: BitStruct():
            value: 0 (BitField(24))
    check_byte: 3 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22
\x86\x80\x00\x00\x00\x00\x0b\x0e\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x03
```

Command 12

Command 12 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(12)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 12 (Command)
        byte_count: 0 (UBInt8)
        payload: Command12_ReadMessageRequest():
        check_byte: 14 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x0c\x00\x0e
# response
hf = HartFrame(12)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 12 (Command)
    byte_count: 26 (UBInt8)
    payload: Command12_ReadMessageResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        message: b'??????????????????????????????????'
    check_byte: 16 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
```

```
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25| 26| 27| 28| 29| 30| 31| 32| 33| 34
\x86\x80\x00\x00\x00\x00\x0c\x1a\x00\x00\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xf
```

Command 13

Command 13 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(13)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 13 (Command)
        byte_count: 0 (UBInt8)
        payload: Command13_ReadTagDescriptorDateRequest():
        check_byte: 15 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x0d\x00\x0f
# response
hf = HartFrame(13)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 13 (Command)
    byte_count: 23 (UBInt8)
    payload: Command13_ReadTagDescriptorDateResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
```

```
        tag: b'????????'
        descriptor: b'????????????????'
        date: datetime.date(1900, 1, 1)
    check_byte: 28 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25| 26| 27| 28| 29| 30| 31
\x86\x80\x00\x00\x00\x00\x0d\x17\x00\x00\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xf
```

Command 14

Command 14 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(14)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 14 (Command)
        byte_count: 0 (UBInt8)
        payload: Command14_ReadPrimaryVariableTransducerInformationRequest():
        check_byte: 12 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x0e\x00\x0c
# response
hf = HartFrame(14)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
```

```
    [
    ]
    command: 14 (Command)
    byte_count: 18 (UBInt8)
    payload: Command14_ReadPrimaryVariableTransducerInformationResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        serial_number: BitStruct():
            value: 0 (BitField(24))
        units_code: 0 (UBInt8)
        upper_limit: 0.0 (BFloat)
        lower_limit: 0.0 (BFloat)
        minimum_span: 0.0 (BFloat)
    check_byte: 26 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25| 26
\x86\x80\x00\x00\x00\x00\x0e\x12\x00\x00\x00\x00
```

```
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 15 (Command)
    byte_count: 20 (UBInt8)
    payload: Command15_ReadDeviceInformationResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        alarm_selection_code: 0 (UBInt8)
        transfer_function_code: 0 (UBInt8)
        units_code: 0 (UBInt8)
        upper_range_value: 0.0 (BFloat)
        lower_range_value: 0.0 (BFloat)
        daming_value: 0.0 (BFloat)
        write_protect_code: 0 (UBInt8)
        reserved: 250 (UBInt8)
        analog_channel_flags: 0 (UBInt8)
    check_byte: 231 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25| 26| 27| 28
\x86\x80\x00\x00\x00\x00\x0f\x14\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
```

Command 16

Command 16 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(16)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 16 (Command)
```

```
        byte_count: 0 (UBInt8)
        payload: Command16_ReadFinalAssemblyNumberRequest():
        check_byte: 18 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x10\x00\x12
# response
hf = HartFrame(16)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 16 (Command)
    byte_count: 5 (UBInt8)
    payload: Command16_ReadFinalAssemblyNumberResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        final_assembly_number: BitStruct():
            value: 0 (BitField(24))
    check_byte: 19 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13
\x86\x80\x00\x00\x00\x00\x10\x05\x00\x00\x00\x00\x00\x13
```

Command 17

Command 17 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(17)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
```

```
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 17 (Command)
        byte_count: 24 (UBInt8)
        payload: Command17_WriteMessageRequest():
            message: b'????????????????????????????????'
        check_byte: 11 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25| 26| 27| 28| 29| 30| 31| 32
\x82\x80\x00\x00\x00\x00\x11\x18\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xf
# response
hf = HartFrame(17)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 17 (Command)
    byte_count: 26 (UBInt8)
    payload: Command17_WriteMessageResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        message: b'??????????????????????????????'
    check_byte: 13 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25| 26| 27| 28| 29| 30| 31| 32| 33| 34
\x86\x80\x00\x00\x00\x00\x11\x1a\x00\x00\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xf
```

Command 18

Command 18 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(18)
print(hf)
    HartFrame():
        delimiter: Delimiter():
```

```
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 18 (Command)
        byte_count: 21 (UBInt8)
        payload: Command18_WriteTageDescriptorDateRequest():
            tag: b'????????'
            descriptor: b'????????????????'
            date: datetime.date(1900, 1, 1)
        check_byte: 5 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25| 26| 27| 28| 29
\x82\x80\x00\x00\x00\x00\x12\x15\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xf
# response
hf = HartFrame(18)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 18 (Command)
    byte_count: 23 (UBInt8)
    payload: Command18_WriteTageDescriptorDateResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        tag: b'????????'
        descriptor: b'????????????????'
        date: datetime.date(1900, 1, 1)
    check_byte: 3 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25| 26| 27| 28| 29| 30| 31
\x86\x80\x00\x00\x00\x00\x12\x17\x00\x00\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xf
```

Command 19

Command 19 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(19)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 19 (Command)
        byte_count: 3 (UBInt8)
        payload: Command19_WriteFinalAssemblyNumberRequest():
            final_assembly_number: BitStruct():
                value: 0 (BitField(24))
        check_byte: 18 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11
\x82\x80\x00\x00\x00\x00\x13\x03\x00\x00\x00\x12
# response
hf = HartFrame(19)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 19 (Command)
    byte_count: 5 (UBInt8)
    payload: Command19_WriteFinalAssemblyNumberResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        final_assembly_number: BitStruct():
            value: 0 (BitField(24))
    check_byte: 16 (CheckByte)
```

```
encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13
\x86\x80\x00\x00\x00\x00\x13\x05\x00\x00\x00\x00\x00\x10
```

Command 20

Command 20 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(20)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 20 (Command)
        byte_count: 0 (UBInt8)
        payload: Command20_ReadLongTagRequest():
        check_byte: 22 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x14\x00\x16
# response
hf = HartFrame(20)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 20 (Command)
    byte_count: 34 (UBInt8)
    payload: Command20_ReadLongTagResponse():
```

```
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        long_tag: b''
    check_byte: 48 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25| 26| 27| 28| 29| 30| 31| 32| 33| 34| 35| 36| 37| 38| 39| 40| 41| 42
\x86\x80\x00\x00\x00\x00\x14\x22\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
```

Command 21

Command 21 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(21)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 21 (Command)
        byte_count: 32 (UBInt8)
        payload: Command21_ReadUniqueIdentifierAssociatedWithLongTagRequest():
            long_tag: b''
        check_byte: 55 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25| 26| 27| 28| 29| 30| 31| 32| 33| 34| 35| 36| 37| 38| 39| 40
\x82\x80\x00\x00\x00\x00\x15\x20\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
# response
hf = HartFrame(21)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
```

```
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 21 (Command)
    byte_count: 14 (UBInt8)
    payload: Command21_ReadUniqueIdentifierAssociatedWithLongTagResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        data_object_id: 0 (UBInt8)
        device_type: 0 (UBInt16)
        minimum_preambles_request: 0 (UBInt8)
        protocol_revision: 0 (UBInt8)
        device_revision_level: 0 (UBInt8)
        software_revision_level: 0 (UBInt8)
        hardware_revision_level_and_physical_signaling_code: 0 (UBInt8)
        flags: 0 (UBInt8)
        device_id: BitStruct():
            value: 0 (BitField(24))
    check_byte: 29 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22
\x86\x80\x00\x00\x00\x00\x15\x0e\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x1d
```

Command 22

Command 22 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(22)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 22 (Command)
        byte_count: 32 (UBInt8)
        payload: Command22_WriteLongTagRequest():
            long_tag: b''
        check_byte: 52 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
```

```
 24| 25| 26| 27| 28| 29| 30| 31| 32| 33| 34| 35| 36| 37| 38| 39| 40
\x82\x80\x00\x00\x00\x00\x16\x20\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
# response
hf = HartFrame(22)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 22 (Command)
    byte_count: 34 (UBInt8)
    payload: Command22_WriteLongTagResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        long_tag: b''
    check_byte: 50 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25| 26| 27| 28| 29| 30| 31| 32| 33| 34| 35| 36| 37| 38| 39| 40| 41| 42
\x86\x80\x00\x00\x00\x00\x16\x22\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
```

Command 38

Command 38 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(38)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 38 (Command)
        byte_count: 0 (UBInt8)
```

```
        payload: Command38_ResetConfigurationChangedFlagRequest():
        check_byte: 36 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x26\x00\x24
# response
hf = HartFrame(38)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 38 (Command)
    byte_count: 4 (UBInt8)
    payload: Command38_ResetConfigurationChangedFlagResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        configuration_change_counter: 0 (UBInt16)
    check_byte: 36 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12
\x86\x80\x00\x00\x00\x00\x26\x04\x00\x00\x00\x00\x24
```

Common Practice Commands

Common Practice Commands

Command 33

Command 33 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(33)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
```

```
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 33 (Command)
        byte_count: 0 (UBInt8)
        payload: Command33_ReadDeviceVariablesRequest():
            device_variable_codes: Array():
            [
            ]
        check_byte: 35 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x21\x00\x23
# response
hf = HartFrame(33)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 33 (Command)
    byte_count: 2 (UBInt8)
    payload: Command33_ReadDeviceVariablesResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        device_variable_codes_units_and_values: Array():
        [
        ]
    check_byte: 37 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10
\x86\x80\x00\x00\x00\x00\x21\x02\x00\x00\x25
```

Command 34

Command 34 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(34)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 34 (Command)
        byte_count: 4 (UBInt8)
        payload: Command34_WritePrimaryVariableDampingValueRequest():
            primary_variable_damping_value: 0.0 (BFloat)
        check_byte: 36 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12
\x82\x80\x00\x00\x00\x00\x22\x04\x00\x00\x00\x00\x24
# response
hf = HartFrame(34)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 34 (Command)
    byte_count: 6 (UBInt8)
    payload: Command34_WritePrimaryVariableDampingValueResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        actual_primary_variable_damping_value: 0.0 (BFloat)
    check_byte: 34 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14
\x86\x80\x00\x00\x00\x00\x22\x06\x00\x00\x00\x00\x00\x00\x22
```

Command 35

Command 35 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(35)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 35 (Command)
        byte_count: 9 (UBInt8)
        payload: Command35_WritePrimaryVariableRangeValuesRequest():
            primary_variable_upper_and_lower_range_value_units_code: 0 (UBInt8)
            primary_variable_upper_range_value: 0.0 (BFloat)
            primary_variable_lower_range_value: 0.0 (BFloat)
        check_byte: 40 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17
\x82\x80\x00\x00\x00\x00\x23\x09\x00\x00\x00\x00\x00\x00\x00\x00\x00\x28
# response
hf = HartFrame(35)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 35 (Command)
    byte_count: 11 (UBInt8)
    payload: Command35_WritePrimaryVariableRangeValuesResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        primary_variable_upper_and_lower_range_value_units_code: 0 (UBInt8)
        primary_variable_upper_range_value: 0.0 (BFloat)
```

```
        primary_variable_lower_range_value: 0.0 (BFloat)
    check_byte: 46 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19
\x86\x80\x00\x00\x00\x00\x23\x0b\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x2e
```

Command 36

Command 36 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(36)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 36 (Command)
        byte_count: 0 (UBInt8)
        payload: Command36_SetPrimaryVariableUpperRangeValueRequest():
        check_byte: 38 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x24\x00\x26
# response
hf = HartFrame(36)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 36 (Command)
```

```
    byte_count: 2 (UBInt8)
    payload: Command36_SetPrimaryVariableUpperRangeValueResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
    check_byte: 32 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10
\x86\x80\x00\x00\x00\x00\x24\x02\x00\x00\x20
```

Command 37

Command 37 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(37)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 37 (Command)
        byte_count: 0 (UBInt8)
        payload: Command37_SetPrimaryVariableLowerRangeValueRequest():
        check_byte: 39 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x25\x00\x27
# response
hf = HartFrame(37)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
```

```
    [
    ]
    command: 37 (Command)
    byte_count: 2 (UBInt8)
    payload: Command37_SetPrimaryVariableLowerRangeValueResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
    check_byte: 33 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10
\x86\x80\x00\x00\x00\x00\x25\x02\x00\x00\x21
```

Command 39

Command 39 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(39)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 39 (Command)
        byte_count: 1 (UBInt8)
        payload: Command39_EEPROMControlRequest():
            eeprom_control_code: 0 (UBInt8)
        check_byte: 36 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x82\x80\x00\x00\x00\x00\x27\x01\x00\x24
# response
hf = HartFrame(39)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
```

```
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 39 (Command)
    byte_count: 3 (UBInt8)
    payload: Command39_EEPROMControlResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        eeprom_control_code: 0 (UBInt8)
    check_byte: 34 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11
\x86\x80\x00\x00\x00\x00\x27\x03\x00\x00\x00\x22
```

Command 40

Command 40 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(40)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 40 (Command)
        byte_count: 4 (UBInt8)
        payload: Command40_EnterExitFixedCurrentModeRequest():
            primary_variable_fixed_current_level: 0.0 (BFloat)
        check_byte: 46 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12
\x82\x80\x00\x00\x00\x00\x28\x04\x00\x00\x00\x00\x2e
# response
hf = HartFrame(40)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
```

```
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 40 (Command)
    byte_count: 6 (UBInt8)
    payload: Command40_EnterExitFixedCurrentModeResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        actual_primary_variable_current_level: 0.0 (BFloat)
    check_byte: 40 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14
\x86\x80\x00\x00\x00\x00\x28\x06\x00\x00\x00\x00\x00\x00\x28
```

Command 41

Command 41 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(41)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 41 (Command)
        byte_count: 0 (UBInt8)
        payload: Command41_PerformSelfTestRequest():
        check_byte: 43 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x29\x00\x2b
# response
hf = HartFrame(41)
```

```
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 41 (Command)
    byte_count: 2 (UBInt8)
    payload: Command41_PerformSelfTestResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
    check_byte: 45 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10
\x86\x80\x00\x00\x00\x00\x29\x02\x00\x00\x2d
```

Command 42

Command 42 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(42)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 42 (Command)
        byte_count: 0 (UBInt8)
        payload: Command42_PerformDeviceResetRequest():
        check_byte: 40 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
```

```
\x82\x80\x00\x00\x00\x00\x2a\x00\x28
# response
hf = HartFrame(42)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 42 (Command)
    byte_count: 2 (UBInt8)
    payload: Command42_PerformDeviceResetResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
    check_byte: 46 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10
\x86\x80\x00\x00\x00\x00\x2a\x02\x00\x00\x2e
```

Command 43

Command 43 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(43)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 43 (Command)
        byte_count: 0 (UBInt8)
        payload: Command43_SetPrimaryVariableZeroRequest():
        check_byte: 41 (CheckByte)
```

```
encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x2b\x00\x29
# response
hf = HartFrame(43)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 43 (Command)
    byte_count: 2 (UBInt8)
    payload: Command43_SetPrimaryVariableZeroResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
    check_byte: 47 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10
\x86\x80\x00\x00\x00\x00\x2b\x02\x00\x00\x2f
```

Command 44

Command 44 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(44)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 44 (Command)
        byte_count: 1 (UBInt8)
```

```
        payload: Command44_WritePrimaryVariableUnitsRequest():
            primary_variable_units_code: 0 (UBInt8)
        check_byte: 47 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x82\x80\x00\x00\x00\x00\x2c\x01\x00\x2f
# response
hf = HartFrame(44)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 44 (Command)
    byte_count: 3 (UBInt8)
    payload: Command44_WritePrimaryVariableUnitsResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        primary_variable_units_code: 0 (UBInt8)
    check_byte: 41 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11
\x86\x80\x00\x00\x00\x00\x2c\x03\x00\x00\x00\x29
```

Command 45

Command 45 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(45)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
```

```
        expansion_bytes: Array():
        [
        ]
        command: 45 (Command)
        byte_count: 4 (UBInt8)
        payload: Command45_TrimLoopCurrentZeroRequest():
            externally_measured_primary_variable_loop_current_level: 0.0 (BFloat)
        check_byte: 43 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12
\x82\x80\x00\x00\x00\x00\x2d\x04\x00\x00\x00\x00\x2b
# response
hf = HartFrame(45)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 45 (Command)
    byte_count: 6 (UBInt8)
    payload: Command45_TrimLoopCurrentZeroResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        actual_measured_primary_variable_loop_current_level: 0.0 (BFloat)
    check_byte: 45 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14
\x86\x80\x00\x00\x00\x00\x2d\x06\x00\x00\x00\x00\x00\x00\x2d
```

Command 46

Command 46 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(46)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
```

```
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 46 (Command)
        byte_count: 4 (UBInt8)
        payload: Command46_TrimLoopCurrentGainRequest():
            externally_measured_primary_variable_loop_current_level: 0.0 (BFloat)
        check_byte: 40 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12
\x82\x80\x00\x00\x00\x00\x2e\x04\x00\x00\x00\x00\x28
# response
hf = HartFrame(46)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 46 (Command)
    byte_count: 6 (UBInt8)
    payload: Command46_TrimLoopCurrentGainResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        actual_measured_primary_variable_loop_current_level: 0.0 (BFloat)
    check_byte: 46 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14
\x86\x80\x00\x00\x00\x00\x2e\x06\x00\x00\x00\x00\x00\x00\x2e
```

Command 47

Command 47 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(47)
print(hf)
```

```
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 47 (Command)
        byte_count: 1 (UBInt8)
        payload: Command47_WritePrimaryVariableTransferFunctionRequest():
            primary_variable_transfer_function_code: 0 (UBInt8)
        check_byte: 44 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x82\x80\x00\x00\x00\x00\x2f\x01\x00\x2c
# response
hf = HartFrame(47)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 47 (Command)
    byte_count: 3 (UBInt8)
    payload: Command47_WritePrimaryVariableTransferFunctionResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        primary_variable_transfer_function_code: 0 (UBInt8)
    check_byte: 42 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11
\x86\x80\x00\x00\x00\x00\x2f\x03\x00\x00\x00\x2a
```

Command 48

Command 48 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(48)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 48 (Command)
        byte_count: 0 (UBInt8)
        payload: Command48_ReadAdditionalDeviceStatusRequest():
        check_byte: 50 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x30\x00\x32
# response
hf = HartFrame(48)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 48 (Command)
    byte_count: 11 (UBInt8)
    payload: Command48_ReadAdditionalDeviceStatusResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        device_specific_status: BitStruct():
            value: 0 (BitField(48))
        extended_device_status: 0 (UBInt8)
        device_operating_mode: 0 (UBInt8)
        status0: 0 (UBInt8)
    check_byte: 61 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
```

```
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19
\x86\x80\x00\x00\x00\x00\x30\x0b\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x3d
```

Command 49

Command 49 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(49)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 49 (Command)
        byte_count: 3 (UBInt8)
        payload: Command49_WritePrimaryVariableTransducerSerialNumberRequest():
            primary_variable_transducer_serial_number: BitStruct():
                value: 0 (BitField(24))
        check_byte: 48 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11
\x82\x80\x00\x00\x00\x00\x31\x03\x00\x00\x00\x30
# response
hf = HartFrame(49)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 49 (Command)
    byte_count: 5 (UBInt8)
    payload: Command49_WritePrimaryVariableTransducerSerialNumberResponse():
        response_code: 'success' (UBInt8)
```

```
        field_device_status:  (FieldDeviceStatus)
        primary_variable_transducer_serial_number: BitStruct():
            value: 0 (BitField(24))
    check_byte: 50 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13
\x86\x80\x00\x00\x00\x00\x31\x05\x00\x00\x00\x00\x00\x32
```

Command 50

Command 50 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(50)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 50 (Command)
        byte_count: 0 (UBInt8)
        payload: Command50_ReadDynamicVariableAssignmentsRequest():
        check_byte: 48 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x32\x00\x30
# response
hf = HartFrame(50)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
```

```
    ]
    command: 50 (Command)
    byte_count: 6 (UBInt8)
    payload: Command50_ReadDynamicVariableAssignmentsResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        primary_variable_device_variable: 0 (UBInt8)
        secondary_variable_device_variable: 0 (UBInt8)
        tertiary_variable_device_variable: 0 (UBInt8)
        quaternary_variable_device_variable: 0 (UBInt8)
    check_byte: 50 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14
\x86\x80\x00\x00\x00\x00\x32\x06\x00\x00\x00\x00\x00\x00\x32
```

Command 51

Command 51 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(51)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 51 (Command)
        byte_count: 0 (UBInt8)
        payload: Command51_WriteDynamicVariableAssignmentsRequest():
            device_variables: Array():
            [
            ]
        check_byte: 49 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x33\x00\x31
# response
hf = HartFrame(51)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
```

```
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 51 (Command)
    byte_count: 2 (UBInt8)
    payload: Command51_WriteDynamicVariableAssignmentsResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        device_variables: Array():
        [
        ]
    check_byte: 55 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10
\x86\x80\x00\x00\x00\x00\x33\x02\x00\x00\x37
```

Command 52

Command 52 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(52)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 52 (Command)
        byte_count: 1 (UBInt8)
        payload: Command52_SetDeviceVariableZeroRequest():
            device_variable_to_zero: 0 (UBInt8)
        check_byte: 55 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
```

```
\x82\x80\x00\x00\x00\x00\x34\x01\x00\x37
# response
hf = HartFrame(52)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 52 (Command)
    byte_count: 3 (UBInt8)
    payload: Command52_SetDeviceVariableZeroResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        device_variable_to_zero: 0 (UBInt8)
    check_byte: 49 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11
\x86\x80\x00\x00\x00\x00\x34\x03\x00\x00\x00\x31
```

Command 53

Command 53 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(53)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 53 (Command)
        byte_count: 2 (UBInt8)
        payload: Command53_WriteDeviceVariableUnitsRequest():
            device_variable_code: 0 (UBInt8)
```

```
            device_variable_units_code: 0 (UBInt8)
        check_byte: 53 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10
\x82\x80\x00\x00\x00\x00\x35\x02\x00\x00\x35
# response
hf = HartFrame(53)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 53 (Command)
    byte_count: 4 (UBInt8)
    payload: Command53_WriteDeviceVariableUnitsResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        device_variable_code: 0 (UBInt8)
        device_variable_units_code: 0 (UBInt8)
    check_byte: 55 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12
\x86\x80\x00\x00\x00\x00\x35\x04\x00\x00\x00\x00\x37
```

Command 54

Command 54 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(54)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
```

```
        expansion_bytes: Array():
        [
        ]
        command: 54 (Command)
        byte_count: 1 (UBInt8)
        payload: Command54_ReadDeviceVariableInformationRequest():
            device_variable_code: 0 (UBInt8)
        check_byte: 53 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x82\x80\x00\x00\x00\x00\x36\x01\x00\x35
# response
hf = HartFrame(54)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 54 (Command)
    byte_count: 30 (UBInt8)
    payload: Command54_ReadDeviceVariableInformationResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        device_variable_code: 0 (UBInt8)
        device_variable_transducer_serial_number: BitStruct():
            value: 0 (BitField(24))
        device_variable_limits: 0 (UBInt8)
        device_variable_upper_transducer_limit: 0.0 (BFloat)
        device_variable_lower_transducer_limit: 0.0 (BFloat)
        device_variable_damping_value: 0.0 (BFloat)
        device_variable_minimum_span: 0.0 (BFloat)
        device_variable_classification: 0 (UBInt8)
        device_variable_family: 0 (UBInt8)
        acquisition_period: 00:00:00 (Time)
        device_variable_properties: 0 (UBInt8)
    check_byte: 46 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25| 26| 27| 28| 29| 30| 31| 32| 33| 34| 35| 36| 37| 38
\x86\x80\x00\x00\x00\x00\x36\x1e\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
```

Command 55

Command 55 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(55)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 55 (Command)
        byte_count: 5 (UBInt8)
        payload: Command55_WriteDeviceVariableDampingValueRequest():
            device_variable_code: 0 (UBInt8)
            device_variable_damping_value: 0.0 (BFloat)
        check_byte: 48 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13
\x82\x80\x00\x00\x00\x00\x37\x05\x00\x00\x00\x00\x00\x30
# response
hf = HartFrame(55)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 55 (Command)
    byte_count: 7 (UBInt8)
    payload: Command55_WriteDeviceVariableDampingValueResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        device_variable_code: 0 (UBInt8)
        device_variable_damping_value: 0.0 (BFloat)
    check_byte: 54 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15
\x86\x80\x00\x00\x00\x00\x37\x07\x00\x00\x00\x00\x00\x00\x00\x36
```

Command 56

Command 56 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(56)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 56 (Command)
        byte_count: 4 (UBInt8)
        payload: Command56_WriteDeviceVariableTransducerSerialNumberRequest():
            device_variable_code: 0 (UBInt8)
            device_variable_transducer_serial_number: BitStruct():
                value: 0 (BitField(24))
        check_byte: 62 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12
\x82\x80\x00\x00\x00\x00\x38\x04\x00\x00\x00\x00\x3e
# response
hf = HartFrame(56)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 56 (Command)
    byte_count: 6 (UBInt8)
    payload: Command56_WriteDeviceVariableTransducerSerialNumberResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        device_variable_code: 0 (UBInt8)
        device_variable_transducer_serial_number: BitStruct():
```

```
            value: 0 (BitField(24))
    check_byte: 56 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14
\x86\x80\x00\x00\x00\x00\x38\x06\x00\x00\x00\x00\x00\x00\x38
```

Command 57

Command 57 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(57)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 57 (Command)
        byte_count: 0 (UBInt8)
        payload: Command57_ReadUnitTagDescriptorDateRequest():
        check_byte: 59 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x39\x00\x3b
# response
hf = HartFrame(57)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 57 (Command)
```

```
    byte_count: 23 (UBInt8)
    payload: Command57_ReadUnitTagDescriptorDateResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        unit_tag: b'????????'
        unit_descriptor: b'??????????????'
        unit_date: BitStruct():
            value: 0 (BitField(24))
    check_byte: 40 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25| 26| 27| 28| 29| 30| 31
\x86\x80\x00\x00\x00\x00\x39\x17\x00\x00\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xf
```

Command 58

Command 58 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(58)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 58 (Command)
        byte_count: 21 (UBInt8)
        payload: Command58_WriteUnitTagDescriptorDateRequest():
            unit_tag: b'????????'
            unit_descriptor: b'??????????????'
            unit_date: BitStruct():
                value: 0 (BitField(24))
        check_byte: 45 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25| 26| 27| 28| 29
\x82\x80\x00\x00\x00\x00\x3a\x15\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xf
# response
hf = HartFrame(58)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
```

```
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 58 (Command)
    byte_count: 23 (UBInt8)
    payload: Command58_WriteUnitTagDescriptorDateResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        unit_tag: b'????????'
        unit_descriptor: b'????????????????'
        unit_date: BitStruct():
            value: 0 (BitField(24))
    check_byte: 43 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25| 26| 27| 28| 29| 30| 31
\x86\x80\x00\x00\x00\x00\x3a\x17\x00\x00\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xf
```

Command 59

Command 59 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(59)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 59 (Command)
        byte_count: 1 (UBInt8)
        payload: Command59_WriteNumberOfResponsePreamblesRequest():
            number_of_preambles: 0 (UBInt8)
        check_byte: 56 (CheckByte)
```

```
encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x82\x80\x00\x00\x00\x00\x3b\x01\x00\x38
# response
hf = HartFrame(59)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 59 (Command)
    byte_count: 3 (UBInt8)
    payload: Command59_WriteNumberOfResponsePreamblesResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        number_of_preambles: 0 (UBInt8)
    check_byte: 62 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11
\x86\x80\x00\x00\x00\x00\x3b\x03\x00\x00\x00\x3e
```

Command 60

Command 60 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(60)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 60 (Command)
```

```
        byte_count: 1 (UBInt8)
        payload: Command60_ReadAnalogChannelAndPercentOfRangeRequest():
            analog_channel_number_code: 0 (UBInt8)
        check_byte: 63 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x82\x80\x00\x00\x00\x00\x3c\x01\x00\x3f
# response
hf = HartFrame(60)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 60 (Command)
    byte_count: 12 (UBInt8)
    payload: Command60_ReadAnalogChannelAndPercentOfRangeResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        analog_channel_number_code: 0 (UBInt8)
        analog_channel_units_code: 0 (UBInt8)
        analog_channel_level: 0.0 (BFloat)
        analog_channel_percent_of_range: 0.0 (BFloat)
    check_byte: 54 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20
\x86\x80\x00\x00\x00\x00\x3c\x0c\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x36
```

Command 61

Command 61 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(61)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
```

```
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 61 (Command)
        byte_count: 0 (UBInt8)
        payload: Command61_ReadDynamicVariablesAndPrimaryVariableAnalogChannelRequest():
        check_byte: 63 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x3d\x00\x3f
# response
hf = HartFrame(61)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 61 (Command)
    byte_count: 7 (UBInt8)
    payload: Command61_ReadDynamicVariablesAndPrimaryVariableAnalogChannelResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        primary_variable_analog_channel_units_code: 0 (UBInt8)
        primary_variable_analog_level: 0.0 (BFloat)
        dynamic_variables: Array():
        [
        ]
    check_byte: 60 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15
\x86\x80\x00\x00\x00\x00\x3d\x07\x00\x00\x00\x00\x00\x00\x00\x3c
```

Command 62

Command 62 example:

```
import oser
from htf.hart import HartFrame
# request
```

```
hf = HartFrame(62)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 62 (Command)
        byte_count: 0 (UBInt8)
        payload: Command62_ReadAnalogChannelsRequest():
            analog_channel_number_codes: Array():
            [
            ]
        check_byte: 60 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x3e\x00\x3c
# response
hf = HartFrame(62)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 62 (Command)
    byte_count: 2 (UBInt8)
    payload: Command62_ReadAnalogChannelsResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        analog_channel_levels: Array():
        [
        ]
    check_byte: 58 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10
\x86\x80\x00\x00\x00\x00\x3e\x02\x00\x00\x3a
```

Command 63

Command 63 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(63)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 63 (Command)
        byte_count: 1 (UBInt8)
        payload: Command63_ReadAnalogChannelInformationRequest():
            analog_channel_number_code: 0 (UBInt8)
        check_byte: 60 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x82\x80\x00\x00\x00\x00\x3f\x01\x00\x3c
# response
hf = HartFrame(63)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 63 (Command)
    byte_count: 19 (UBInt8)
    payload: Command63_ReadAnalogChannelInformationResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        analog_channel_number_code: 0 (UBInt8)
        analog_channel_alarm_selection_code: 0 (UBInt8)
        analog_channel_transfer_function_code: 0 (UBInt8)
        analog_channel_upper_and_lower_range_value_units_code: 0 (UBInt8)
```

```
        analog_channel_upper_range_value: 0.0 (BFloat)
        analog_channel_lower_range_value: 0.0 (BFloat)
        analog_channel_damping_value: 0.0 (BFloat)
        analog_channel_flags: 0 (UBInt8)
    check_byte: 42 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25| 26| 27
\x86\x80\x00\x00\x00\x00\x3f\x13\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
```

Command 64

Command 64 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(64)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 64 (Command)
        byte_count: 5 (UBInt8)
        payload: Command64_WriteAnalogChannelAdditionalDampingValueRequest():
            analog_channel_number_code: 0 (UBInt8)
            analog_channel_additional_damping_value: 0.0 (BFloat)
        check_byte: 71 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13
\x82\x80\x00\x00\x00\x00\x40\x05\x00\x00\x00\x00\x00\x47
# response
hf = HartFrame(64)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
```

```
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 64 (Command)
    byte_count: 7 (UBInt8)
    payload: Command64_WriteAnalogChannelAdditionalDampingValueResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        analog_channel_number_code: 0 (UBInt8)
        analog_channel_additional_damping_value: 0.0 (BFloat)
    check_byte: 65 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15
\x86\x80\x00\x00\x00\x00\x40\x07\x00\x00\x00\x00\x00\x00\x00\x41
```

Command 65

Command 65 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(65)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 65 (Command)
        byte_count: 10 (UBInt8)
        payload: Command65_WriteAnalogChannelRangeValuesRequest():
            analog_channel_number_code: 0 (UBInt8)
            analog_channel_upper_and_lower_range_value_units_code: 0 (UBInt8)
            analog_channel_upper_range_value: 0.0 (BFloat)
            analog_channel_lower_range_value: 0.0 (BFloat)
        check_byte: 73 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18
\x82\x80\x00\x00\x00\x00\x41\x0a\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x49
# response
hf = HartFrame(65)
hf.delimiter.frame_type.set("ack")
```

```
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 65 (Command)
    byte_count: 12 (UBInt8)
    payload: Command65_WriteAnalogChannelRangeValuesResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        analog_channel_number_code: 0 (UBInt8)
        analog_channel_upper_and_lower_range_value_units_code: 0 (UBInt8)
        analog_channel_upper_range_value: 0.0 (BFloat)
        analog_channel_lower_range_value: 0.0 (BFloat)
    check_byte: 75 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20
\x86\x80\x00\x00\x00\x00\x41\x0c\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x4b
```

Command 66

Command 66 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(66)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 66 (Command)
        byte_count: 6 (UBInt8)
        payload: Command66_EnterExitFixedAnalogChannelModeRequest():
            analog_channel_number_code: 0 (UBInt8)
            analog_channel_units_code: 0 (UBInt8)
```

```
            fixed_analog_channel_level: 0.0 (BFloat)
        check_byte: 70 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14
\x82\x80\x00\x00\x00\x00\x42\x06\x00\x00\x00\x00\x00\x00\x46
# response
hf = HartFrame(66)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 66 (Command)
    byte_count: 8 (UBInt8)
    payload: Command66_EnterExitFixedAnalogChannelModeResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        analog_channel_number_code: 0 (UBInt8)
        analog_channel_units_code: 0 (UBInt8)
        fixed_analog_channel_level: 0.0 (BFloat)
    check_byte: 76 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16
\x86\x80\x00\x00\x00\x00\x42\x08\x00\x00\x00\x00\x00\x00\x00\x00\x4c
```

Command 67

Command 67 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(67)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
```

```
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 67 (Command)
        byte_count: 6 (UBInt8)
        payload: Command67_TrimAnalogChannelZeroRequest():
            analog_channel_number_code: 0 (UBInt8)
            analog_channel_units_code: 0 (UBInt8)
            externally_measured_analog_channel_level: 0.0 (BFloat)
        check_byte: 71 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14
\x82\x80\x00\x00\x00\x00\x43\x06\x00\x00\x00\x00\x00\x00\x47
# response
hf = HartFrame(67)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 67 (Command)
    byte_count: 8 (UBInt8)
    payload: Command67_TrimAnalogChannelZeroResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        analog_channel_number_code: 0 (UBInt8)
        analog_channel_units_code: 0 (UBInt8)
        actual_channel_level: 0.0 (BFloat)
    check_byte: 77 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16
\x86\x80\x00\x00\x00\x00\x43\x08\x00\x00\x00\x00\x00\x00\x00\x00\x4d
```

Command 68

Command 68 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(68)
print(hf)
```

```
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 68 (Command)
        byte_count: 6 (UBInt8)
        payload: Command68_TrimAnalogChannelGainRequest():
            analog_channel_number_code: 0 (UBInt8)
            analog_channel_units_code: 0 (UBInt8)
            externally_measured_analog_channel_level: 0.0 (BFloat)
        check_byte: 64 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14
\x82\x80\x00\x00\x00\x00\x44\x06\x00\x00\x00\x00\x00\x00\x40
# response
hf = HartFrame(68)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 68 (Command)
    byte_count: 8 (UBInt8)
    payload: Command68_TrimAnalogChannelGainResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        analog_channel_number_code: 0 (UBInt8)
        analog_channel_units_code: 0 (UBInt8)
        externally_measured_analog_channel_level: 0.0 (BFloat)
    check_byte: 74 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16
\x86\x80\x00\x00\x00\x00\x44\x08\x00\x00\x00\x00\x00\x00\x00\x00\x4a
```

Command 69

Command 69 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(69)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 69 (Command)
        byte_count: 2 (UBInt8)
        payload: Command69_WriteAnalogChannelTransferFunctionRequest():
            analog_channel_number_code: 0 (UBInt8)
            analog_channel_transfer_function_code: 0 (UBInt8)
        check_byte: 69 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10
\x82\x80\x00\x00\x00\x00\x45\x02\x00\x00\x45
# response
hf = HartFrame(69)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 69 (Command)
    byte_count: 4 (UBInt8)
    payload: Command69_WriteAnalogChannelTransferFunctionResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        analog_channel_number_code: 0 (UBInt8)
        analog_channel_transfer_function_code: 0 (UBInt8)
    check_byte: 71 (CheckByte)
```

```
encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12
\x86\x80\x00\x00\x00\x00\x45\x04\x00\x00\x00\x00\x47
```

Command 70

Command 70 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(70)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 70 (Command)
        byte_count: 1 (UBInt8)
        payload: Command70_ReadAnalogChannelEndpointValuesRequest():
            analog_channel_number_code: 0 (UBInt8)
        check_byte: 69 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x82\x80\x00\x00\x00\x00\x46\x01\x00\x45
# response
hf = HartFrame(70)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 70 (Command)
    byte_count: 20 (UBInt8)
```

```
    payload: Command70_ReadAnalogChannelEndpointValuesResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        analog_channel_number_code: 0 (UBInt8)
        analog_channel_upper_and_lower_range_value_units_code: 0 (UBInt8)
        analog_channel_upper_endpoint_value: 0.0 (BFloat)
        analog_channel_lower_endpoint_value: 0.0 (BFloat)
        analog_channel_upper_limit_value: 0.0 (BFloat)
        analog_channel_lower_limit_value: 0.0 (BFloat)
    check_byte: 84 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25| 26| 27| 28
\x86\x80\x00\x00\x00\x00\x46\x14\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
```

Command 71

Command 71 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(71)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 71 (Command)
        byte_count: 1 (UBInt8)
        payload: Command71_LockDeviceRequest():
            lock_code: 0 (UBInt8)
        check_byte: 68 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x82\x80\x00\x00\x00\x00\x47\x01\x00\x44
# response
hf = HartFrame(71)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
```

```
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 71 (Command)
    byte_count: 3 (UBInt8)
    payload: Command71_LockDeviceResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        lock_code: 0 (UBInt8)
    check_byte: 66 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11
\x86\x80\x00\x00\x00\x00\x47\x03\x00\x00\x00\x42
```

Command 72

Command 72 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(72)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 72 (Command)
        byte_count: 1 (UBInt8)
        payload: Command72_SquawkRequest():
            squawk_control: 0 (UBInt8)
        check_byte: 75 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x82\x80\x00\x00\x00\x00\x48\x01\x00\x4b
# response
hf = HartFrame(72)
hf.delimiter.frame_type.set("ack")
```

```
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 72 (Command)
    byte_count: 3 (UBInt8)
    payload: Command72_SquawkResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        squawk_control: 0 (UBInt8)
    check_byte: 77 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11
\x86\x80\x00\x00\x00\x00\x48\x03\x00\x00\x00\x4d
```

Command 73

Command 73 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(73)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 73 (Command)
        byte_count: 0 (UBInt8)
        payload: Command73_FindDeviceRequest():
        check_byte: 75 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
```

```
\x82\x80\x00\x00\x00\x00\x49\x00\x4b
# response
hf = HartFrame(73)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 73 (Command)
    byte_count: 14 (UBInt8)
    payload: Command73_FindDeviceResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        data_object_id: 0 (UBInt8)
        device_type: 0 (UBInt16)
        minimum_preambles_request: 0 (UBInt8)
        protocol_revision: 0 (UBInt8)
        device_revision_level: 0 (UBInt8)
        software_revision_level: 0 (UBInt8)
        hardware_revision_level_and_physical_signaling_code: 0 (UBInt8)
        flags: 0 (UBInt8)
        device_id: BitStruct():
            value: 0 (BitField(24))
    check_byte: 65 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22
\x86\x80\x00\x00\x00\x00\x49\x0e\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x41
```

Command 74

Command 74 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(74)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
```

```
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 74 (Command)
        byte_count: 0 (UBInt8)
        payload: Command74_ReadIOSystemCapabilitiesRequest():
        check_byte: 72 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x4a\x00\x48
# response
hf = HartFrame(74)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 74 (Command)
    byte_count: 10 (UBInt8)
    payload: Command74_ReadIOSystemCapabilitiesResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        maximum_number_of_io_cards: 0 (UBInt8)
        maximum_number_of_channels_per_card: 0 (UBInt8)
        maximum_number_of_subdevices_per_channel: 0 (UBInt8)
        number_of_devices: 0 (UBInt16)
        maximum_number_of_delayed_responses: 0 (UBInt8)
        master_mode: 0 (UBInt8)
        retry_count: 0 (UBInt8)
    check_byte: 70 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18
\x86\x80\x00\x00\x00\x00\x4a\x0a\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x46
```

Command 75

Command 75 example:

```
import oser
from htf.hart import HartFrame
# request
```

```
hf = HartFrame(75)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 75 (Command)
        byte_count: 3 (UBInt8)
        payload: Command75_PollSubDeviceRequest():
            io_card: 0 (UBInt8)
            channel: 0 (UBInt8)
            subdevice_polling_address: 0 (UBInt8)
        check_byte: 74 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11
\x82\x80\x00\x00\x00\x00\x4b\x03\x00\x00\x00\x4a
# response
hf = HartFrame(75)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 75 (Command)
    byte_count: 14 (UBInt8)
    payload: Command75_PollSubDeviceResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        data_object_id: 0 (UBInt8)
        device_type: 0 (UBInt16)
        minimum_preambles_request: 0 (UBInt8)
        protocol_revision: 0 (UBInt8)
        device_revision_level: 0 (UBInt8)
        software_revision_level: 0 (UBInt8)
        hardware_revision_level_and_physical_signaling_code: 0 (UBInt8)
        flags: 0 (UBInt8)
        device_id: BitStruct():
            value: 0 (BitField(24))
    check_byte: 67 (CheckByte)
```

```
encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22
\x86\x80\x00\x00\x00\x00\x4b\x0e\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x43
```

Command 76

Command 76 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(76)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 76 (Command)
        byte_count: 0 (UBInt8)
        payload: Command76_ReadLockDeviceStateRequest():
        check_byte: 78 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x4c\x00\x4e
# response
hf = HartFrame(76)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 76 (Command)
    byte_count: 3 (UBInt8)
    payload: Command76_ReadLockDeviceStateResponse():
```

```
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        lock_status: 0 (UBInt8)
    check_byte: 73 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11
\x86\x80\x00\x00\x00\x00\x4c\x03\x00\x00\x00\x49
```

Command 77

Command 77 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(77)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 77 (Command)
        byte_count: 11 (UBInt8)
        payload: Command77_SendCommandToSubDeviceRequest():
            io_card: 0 (UBInt8)
            channel: 0 (UBInt8)
            transmit_preamble_count: 0 (UBInt8)
            delimiter: Delimiter():
                address_type: 'unique' (Flag)
                number_of_expansion_bytes: 0 (BitField(2))
                physical_layer_type: 'asynchronous' (BitField(2))
                frame_type: 'stx' (BitField(3))
            address: Array():
            [
                @0: 0 (UBInt8)
                @1: 0 (UBInt8)
                @2: 0 (UBInt8)
                @3: 0 (UBInt8)
                @4: 0 (UBInt8)
            ]
            command: 0 (UBInt8)
            byte_count: 0 (UBInt8)
            data: Array():
            [
            ]
        check_byte: 198 (CheckByte)
```

```
encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19
\x82\x80\x00\x00\x00\x00\x4d\x0b\x00\x00\x00\x82\x00\x00\x00\x00\x00\x00\x00\xc6
# response
hf = HartFrame(77)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 77 (Command)
    byte_count: 12 (UBInt8)
    payload: Command77_SendCommandToSubDeviceResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        io_card: 0 (UBInt8)
        channel: 0 (UBInt8)
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Array():
        [
            @0: 0 (UBInt8)
            @1: 0 (UBInt8)
            @2: 0 (UBInt8)
            @3: 0 (UBInt8)
            @4: 0 (UBInt8)
        ]
        command: 0 (UBInt8)
        byte_count: 0 (UBInt8)
        data: Array():
        [
        ]
    check_byte: 197 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20
\x86\x80\x00\x00\x00\x00\x4d\x0c\x00\x00\x00\x00\x82\x00\x00\x00\x00\x00\x00\x00\xc5
```

Command 78

Command 78 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(78)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 78 (Command)
        byte_count: 1 (UBInt8)
        payload: Command78_ReadAggregatedCommandsRequest():
            requested_commands: 0 (UBInt8)
            commands: Array():
            [
            ]
        check_byte: 77 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x82\x80\x00\x00\x00\x00\x4e\x01\x00\x4d
# response
hf = HartFrame(78)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 78 (Command)
    byte_count: 4 (UBInt8)
    payload: Command78_ReadAggregatedCommandsResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        extended_field_device_status: 0 (UBInt8)
        requested_commands: 0 (UBInt8)
        commands: Array():
        [
        ]
    check_byte: 76 (CheckByte)
```

```
encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12
\x86\x80\x00\x00\x00\x00\x4e\x04\x00\x00\x00\x00\x4c
```

Command 79

Command 79 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(79)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 79 (Command)
        byte_count: 8 (UBInt8)
        payload: Command79_WriteDeviceVariableRequest():
            device_variable_code: 0 (UBInt8)
            write_device_variable_command_code: 0 (UBInt8)
            units_code: 0 (UBInt8)
            device_variable_value: 0.0 (BFloat)
            device_variable_status: 0 (UBInt8)
        check_byte: 69 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16
\x82\x80\x00\x00\x00\x00\x4f\x08\x00\x00\x00\x00\x00\x00\x00\x00\x45
# response
hf = HartFrame(79)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
```

```
    [
    ]
    command: 79 (Command)
    byte_count: 10 (UBInt8)
    payload: Command79_WriteDeviceVariableResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        device_variable_code: 0 (UBInt8)
        write_device_variable_command_code: 0 (UBInt8)
        units_code: 0 (UBInt8)
        device_variable_value: 0.0 (BFloat)
        device_variable_status: 0 (UBInt8)
    check_byte: 67 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18
\x86\x80\x00\x00\x00\x00\x4f\x0a\x00\x00\x00\x00\x00\x00\x00\x00\x00\x43
```

Command 80

Command 80 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(80)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 80 (Command)
        byte_count: 1 (UBInt8)
        payload: Command80_ReadDeviceVariableTrimPointsRequest():
            device_variable_code: 0 (UBInt8)
        check_byte: 83 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x82\x80\x00\x00\x00\x00\x50\x01\x00\x53
# response
hf = HartFrame(80)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
```

```
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 80 (Command)
    byte_count: 12 (UBInt8)
    payload: Command80_ReadDeviceVariableTrimPointsResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        device_variable_code: 0 (UBInt8)
        trim_points_units_code: 0 (UBInt8)
        lower_trim_point: 0.0 (BFloat)
        upper_trim_point: 0.0 (BFloat)
    check_byte: 90 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20
\x86\x80\x00\x00\x00\x00\x50\x0c\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x5a
```

Command 81

Command 81 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(81)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 81 (Command)
        byte_count: 1 (UBInt8)
        payload: Command81_ReadDeviceVariableTrimGuidelinesRequest():
            device_variable_guidelines: 0 (UBInt8)
        check_byte: 82 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
```

```
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x82\x80\x00\x00\x00\x00\x51\x01\x00\x52
# response
hf = HartFrame(81)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 81 (Command)
    byte_count: 25 (UBInt8)
    payload: Command81_ReadDeviceVariableTrimGuidelinesResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        device_variable_guidelines: 0 (UBInt8)
        trim_points_supported: 0 (UBInt8)
        trim_points_units_code: 0 (UBInt8)
        minimum_lower_trim_point_value: 0.0 (BFloat)
        maximum_lower_trim_point_value: 0.0 (BFloat)
        minimum_upper_trim_point_value: 0.0 (BFloat)
        maximum_upper_trim_point_value: 0.0 (BFloat)
        minimum_differential: 0.0 (BFloat)
    check_byte: 78 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25| 26| 27| 28| 29| 30| 31| 32| 33
\x86\x80\x00\x00\x00\x00\x51\x19\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
```

Command 82

Command 82 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(82)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
```

```
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 82 (Command)
        byte_count: 7 (UBInt8)
        payload: Command82_WriteDeviceVariableTrimPointRequest():
            device_variable_code: 0 (UBInt8)
            trim_point: 0 (UBInt8)
            trim_point_units_code: 0 (UBInt8)
            trim_point_value: 0.0 (BFloat)
        check_byte: 87 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15
\x82\x80\x00\x00\x00\x00\x52\x07\x00\x00\x00\x00\x00\x00\x00\x57
# response
hf = HartFrame(82)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 82 (Command)
    byte_count: 9 (UBInt8)
    payload: Command82_WriteDeviceVariableTrimPointResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        device_variable_code: 0 (UBInt8)
        trim_point: 0 (UBInt8)
        trim_point_units_code: 0 (UBInt8)
        trim_point_value: 0.0 (BFloat)
    check_byte: 93 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17
\x86\x80\x00\x00\x00\x00\x52\x09\x00\x00\x00\x00\x00\x00\x00\x00\x00\x5d
```

Command 83

Command 83 example:

```
import oser
from htf.hart import HartFrame
```

```
# request
hf = HartFrame(83)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 83 (Command)
        byte_count: 1 (UBInt8)
        payload: Command83_ResetDeviceVariableTrimRequest():
            device_variable_code: 0 (UBInt8)
        check_byte: 80 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x82\x80\x00\x00\x00\x00\x53\x01\x00\x50
# response
hf = HartFrame(83)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 83 (Command)
    byte_count: 3 (UBInt8)
    payload: Command83_ResetDeviceVariableTrimResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        device_variable_code: 0 (UBInt8)
    check_byte: 86 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11
\x86\x80\x00\x00\x00\x00\x53\x03\x00\x00\x00\x56
```

Command 84

Command 84 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(84)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 84 (Command)
        byte_count: 2 (UBInt8)
        payload: Command84_ReadSubDeviceIdentitySummaryRequest():
            subdevice_index: 0 (UBInt16)
        check_byte: 84 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10
\x82\x80\x00\x00\x00\x00\x54\x02\x00\x00\x54
# response
hf = HartFrame(84)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 84 (Command)
    byte_count: 50 (UBInt8)
    payload: Command84_ReadSubDeviceIdentitySummaryResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        subdevice_index: 0 (UBInt16)
        io_card: 0 (UBInt8)
        channel: 0 (UBInt8)
        manufacturer_id: 0 (UBInt16)
```

```
        expanded_device_type_code: 0 (UBInt16)
        device_id: BitStruct():
            value: 0 (BitField(24))
        universal_command_revision_level: 0 (UBInt8)
        long_tag: b''
        device_revision: 0 (UBInt8)
        device_profile: 0 (UBInt8)
        private_label_distributor_code: 0 (UBInt16)
    check_byte: 96 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25| 26| 27| 28| 29| 30| 31| 32| 33| 34| 35| 36| 37| 38| 39| 40| 41| 42| 43| 44| 45| 46| 47| 48|
49| 50| 51| 52| 53| 54| 55| 56| 57| 58
\x86\x80\x00\x00\x00\x00\x54\x32\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
```

Command 85

Command 85 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(85)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 85 (Command)
        byte_count: 2 (UBInt8)
        payload: Command85_ReadIOChannelStatisticsRequest():
            io_card: 0 (UBInt8)
            channel: 0 (UBInt8)
        check_byte: 85 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10
\x82\x80\x00\x00\x00\x00\x55\x02\x00\x00\x55
# response
hf = HartFrame(85)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
```

```
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 85 (Command)
    byte_count: 14 (UBInt8)
    payload: Command85_ReadIOChannelStatisticsResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        io_card: 0 (UBInt8)
        channel: 0 (UBInt8)
        stx_count: 0 (UBInt16)
        ack_count: 0 (UBInt16)
        ostx_count: 0 (UBInt16)
        oack_count: 0 (UBInt16)
        back_count: 0 (UBInt16)
    check_byte: 93 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22
\x86\x80\x00\x00\x00\x00\x55\x0e\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x5d
```

Command 86

Command 86 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(86)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 86 (Command)
        byte_count: 2 (UBInt8)
        payload: Command86_ReadSubDeviceStatisticsRequest():
            subdevice_index: 0 (UBInt16)
        check_byte: 86 (CheckByte)
```

```
encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10
\x82\x80\x00\x00\x00\x56\x02\x00\x00\x56
# response
hf = HartFrame(86)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 86 (Command)
    byte_count: 10 (UBInt8)
    payload: Command86_ReadSubDeviceStatisticsResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        subdevice_index: 0 (UBInt16)
        stx_count: 0 (UBInt16)
        ack_count: 0 (UBInt16)
        back_count: 0 (UBInt16)
    check_byte: 90 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18
\x86\x80\x00\x00\x00\x00\x56\x0a\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x5a
```

Command 87

Command 87 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(87)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
```

```
        [
        ]
        command: 87 (Command)
        byte_count: 1 (UBInt8)
        payload: Command87_WriteIOSystemMasterModeRequest():
            master_mode: 0 (UBInt8)
        check_byte: 84 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x82\x80\x00\x00\x00\x00\x57\x01\x00\x54
# response
hf = HartFrame(87)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 87 (Command)
    byte_count: 3 (UBInt8)
    payload: Command87_WriteIOSystemMasterModeResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        master_mode: 0 (UBInt8)
    check_byte: 82 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11
\x86\x80\x00\x00\x00\x00\x57\x03\x00\x00\x00\x52
```

Command 88

Command 88 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(88)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
```

```
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 88 (Command)
        byte_count: 1 (UBInt8)
        payload: Command88_WriteIOSystemRetryCountRequest():
            retry_count: 0 (UBInt8)
        check_byte: 91 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x82\x80\x00\x00\x00\x00\x58\x01\x00\x5b
# response
hf = HartFrame(88)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 88 (Command)
    byte_count: 3 (UBInt8)
    payload: Command88_WriteIOSystemRetryCountResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        retry_count: 0 (UBInt8)
    check_byte: 93 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11
\x86\x80\x00\x00\x00\x00\x58\x03\x00\x00\x00\x5d
```

Command 89

Command 89 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(89)
print(hf)
    HartFrame():
```

```
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 89 (Command)
        byte_count: 10 (UBInt8)
        payload: Command89_SetRealTimeClockRequest():
            time_set_code: 0 (UBInt8)
            date: datetime.date(1900, 1, 1)
            time: 00:00:00 (Time)
            padding: 0 (UBInt16)
        check_byte: 81 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18
\x82\x80\x00\x00\x00\x00\x59\x0a\x00\x01\x01\x00\x00\x00\x00\x00\x00\x00\x51
# response
hf = HartFrame(89)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 89 (Command)
    byte_count: 10 (UBInt8)
    payload: Command89_SetRealTimeClockResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        time_set_code: 0 (UBInt8)
        date: datetime.date(1900, 1, 1)
        time: 00:00:00 (Time)
    check_byte: 85 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18
\x86\x80\x00\x00\x00\x00\x59\x0a\x00\x00\x00\x01\x01\x00\x00\x00\x00\x00\x55
```

Command 90

Command 90 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(90)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 90 (Command)
        byte_count: 0 (UBInt8)
        payload: Command90_ReadRealTimeClockRequest():
        check_byte: 88 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x5a\x00\x58
# response
hf = HartFrame(90)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 90 (Command)
    byte_count: 17 (UBInt8)
    payload: Command90_ReadRealTimeClockResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        date: datetime.date(1900, 1, 1)
        time: 00:00:00 (Time)
        date_clock_last_set: datetime.date(1900, 1, 1)
        time_clock_last_set: 00:00:00 (Time)
        rtc_flags: 0 (UBInt8)
```

```
    check_byte: 77 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25
\x86\x80\x00\x00\x00\x00\x5a\x11\x00\x00\x01\x01\x00\x00\x00\x00\x00\x01\x01\x00\x00\x00\x00\x00\x00\x4
```

Command 91

Command 91 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(91)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 91 (Command)
        byte_count: 1 (UBInt8)
        payload: Command91_ReadTrendConfigurationRequest():
            trend_number: 0 (UBInt8)
        check_byte: 88 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x82\x80\x00\x00\x00\x00\x5b\x01\x00\x58
# response
hf = HartFrame(91)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
```

```
    command: 91 (Command)
    byte_count: 10 (UBInt8)
    payload: Command91_ReadTrendConfigurationResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        trend_number: 0 (UBInt8)
        total_trends_supported: 0 (UBInt8)
        trend_control_code: 0 (UBInt8)
        device_variable_code: 0 (UBInt8)
        trend_sample_interval: 00:00:00 (Time)
    check_byte: 87 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18
\x86\x80\x00\x00\x00\x00\x5b\x0a\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x57
```

Command 92

Command 92 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(92)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 92 (Command)
        byte_count: 7 (UBInt8)
        payload: Command92_WriteTrendConfigurationRequest():
            trend_number: 0 (UBInt8)
            trend_control_code: 0 (UBInt8)
            device_variable_code: 0 (UBInt8)
            trend_sample_interval: 00:00:00 (Time)
        check_byte: 89 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15
\x82\x80\x00\x00\x00\x00\x5c\x07\x00\x00\x00\x00\x00\x00\x00\x59
# response
hf = HartFrame(92)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
```

```
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 92 (Command)
    byte_count: 9 (UBInt8)
    payload: Command92_WriteTrendConfigurationResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        trend_number: 0 (UBInt8)
        trend_control_code: 0 (UBInt8)
        device_variable_code: 0 (UBInt8)
        trend_sample_interval: 00:00:00 (Time)
    check_byte: 83 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17
\x86\x80\x00\x00\x00\x00\x5c\x09\x00\x00\x00\x00\x00\x00\x00\x00\x00\x53
```

Command 93

Command 93 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(93)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 93 (Command)
        byte_count: 1 (UBInt8)
        payload: Command93_ReadTrendRequest():
            trend_number: 0 (UBInt8)
        check_byte: 94 (CheckByte)

encoded = hf.encode()
```

```
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x82\x80\x00\x00\x00\x00\x5d\x01\x00\x5e
# response
hf = HartFrame(93)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 93 (Command)
    byte_count: 77 (UBInt8)
    payload: Command93_ReadTrendResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        trend_number: 0 (UBInt8)
        device_variable_code: 0 (UBInt8)
        device_variable_classification: 0 (UBInt8)
        device_variable_units_code: 0 (UBInt8)
        date_trend_value0: datetime.date(1900, 1, 1)
        time_trend_value0: 00:00:00 (Time)
        time_sample_interval: 00:00:00 (Time)
        trend_values: Array():
        [
            @0: TrendValueAndStatus():
                trend_value: 0.0 (BFloat)
                trend_status: 0 (UBInt8)
            @1: TrendValueAndStatus():
                trend_value: 0.0 (BFloat)
                trend_status: 0 (UBInt8)
            @2: TrendValueAndStatus():
                trend_value: 0.0 (BFloat)
                trend_status: 0 (UBInt8)
            @3: TrendValueAndStatus():
                trend_value: 0.0 (BFloat)
                trend_status: 0 (UBInt8)
            @4: TrendValueAndStatus():
                trend_value: 0.0 (BFloat)
                trend_status: 0 (UBInt8)
            @5: TrendValueAndStatus():
                trend_value: 0.0 (BFloat)
                trend_status: 0 (UBInt8)
            @6: TrendValueAndStatus():
                trend_value: 0.0 (BFloat)
                trend_status: 0 (UBInt8)
            @7: TrendValueAndStatus():
                trend_value: 0.0 (BFloat)
                trend_status: 0 (UBInt8)
            @8: TrendValueAndStatus():
                trend_value: 0.0 (BFloat)
```

```
                trend_status: 0 (UBInt8)
            @9: TrendValueAndStatus():
                trend_value: 0.0 (BFloat)
                trend_status: 0 (UBInt8)
            @10: TrendValueAndStatus():
                trend_value: 0.0 (BFloat)
                trend_status: 0 (UBInt8)
            @11: TrendValueAndStatus():
                trend_value: 0.0 (BFloat)
                trend_status: 0 (UBInt8)
        ]
    check_byte: 22 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25| 26| 27| 28| 29| 30| 31| 32| 33| 34| 35| 36| 37| 38| 39| 40| 41| 42| 43| 44| 45| 46| 47| 48|
49| 50| 51| 52| 53| 54| 55| 56| 57| 58| 59| 60| 61| 62| 63| 64| 65| 66| 67| 68| 69| 70| 71| 72| 73|
74| 75| 76| 77| 78| 79| 80| 81| 82| 83| 84| 85
\x86\x80\x00\x00\x00\x00\x5d\x4d\x00\x00\x00\x00\x00\x00\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x0
```

Command 94

Command 94 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(94)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 94 (Command)
        byte_count: 0 (UBInt8)
        payload: Command94_ReadIOSystemClientSideCommunicationStatisticsRequest():
        check_byte: 92 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x5e\x00\x5c
# response
hf = HartFrame(94)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
```

```
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 94 (Command)
    byte_count: 18 (UBInt8)
    payload: Command94_ReadIOSystemClientSideCommunicationStatisticsResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        messages_received: 0 (UBInt32)
        messages_returned: 0 (UBInt32)
        requests_forwared: 0 (UBInt32)
        responses_returned: 0 (UBInt32)
    check_byte: 74 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25| 26
\x86\x80\x00\x00\x00\x00\x5e\x12\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
```

Command 95

Command 95 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(95)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 95 (Command)
        byte_count: 0 (UBInt8)
        payload: Command95_ReadDeviceCommunicationStatisticsRequest():
        check_byte: 93 (CheckByte)

encoded = hf.encode()
```

```
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x5f\x00\x5d
# response
hf = HartFrame(95)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 95 (Command)
    byte_count: 8 (UBInt8)
    payload: Command95_ReadDeviceCommunicationStatisticsResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        stx_count: 0 (UBInt16)
        ack_count: 0 (UBInt16)
        back_count: 0 (UBInt16)
    check_byte: 81 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16
\x86\x80\x00\x00\x00\x00\x5f\x08\x00\x00\x00\x00\x00\x00\x00\x00\x51
```

Command 96

Command 96 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(96)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
```

```
        command: 96 (Command)
        byte_count: 1 (UBInt8)
        payload: Command96_ReadSynchronousActionRequest():
            action_number: 0 (UBInt8)
        check_byte: 99 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x82\x80\x00\x00\x00\x00\x60\x01\x00\x63
# response
hf = HartFrame(96)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 96 (Command)
    byte_count: 15 (UBInt8)
    payload: Command96_ReadSynchronousActionResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        action_number: 0 (UBInt8)
        total_action: 0 (UBInt8)
        action_control: 0 (UBInt8)
        device_variable_code: 0 (UBInt8)
        command: 0 (UBInt16)
        trigger_date: datetime.date(1900, 1, 1)
        trigger_time: 00:00:00 (Time)
    check_byte: 105 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23
\x86\x80\x00\x00\x00\x00\x60\x0f\x00\x00\x00\x00\x00\x00\x00\x00\x01\x01\x00\x00\x00\x00\x00\x69
```

Command 97

Command 97 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(97)
print(hf)
    HartFrame():
        delimiter: Delimiter():
```

```
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 97 (Command)
        byte_count: 12 (UBInt8)
        payload: Command97_ConfigureSynchronousActionRequest():
            action_number: 0 (UBInt8)
            action_control: 0 (UBInt8)
            device_variable_code: 0 (UBInt8)
            command: 0 (UBInt16)
            trigger_date: datetime.date(1900, 1, 1)
            trigger_time: 00:00:00 (Time)
        check_byte: 111 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20
\x82\x80\x00\x00\x00\x00\x61\x0c\x00\x00\x00\x00\x00\x01\x01\x00\x00\x00\x00\x00\x6f
# response
hf = HartFrame(97)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 97 (Command)
    byte_count: 14 (UBInt8)
    payload: Command97_ConfigureSynchronousActionResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        action_number: 0 (UBInt8)
        action_control: 0 (UBInt8)
        device_variable_code: 0 (UBInt8)
        command: 0 (UBInt16)
        trigger_date: datetime.date(1900, 1, 1)
        trigger_time: 00:00:00 (Time)
    check_byte: 105 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22
\x86\x80\x00\x00\x00\x00\x61\x0e\x00\x00\x00\x00\x00\x00\x00\x01\x01\x00\x00\x00\x00\x00\x69
```

Command 98

Command 98 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(98)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 98 (Command)
        byte_count: 1 (UBInt8)
        payload: Command98_ReadCommandActionRequest():
            action_number: 0 (UBInt8)
        check_byte: 97 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x82\x80\x00\x00\x00\x00\x62\x01\x00\x61
# response
hf = HartFrame(98)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 98 (Command)
    byte_count: 6 (UBInt8)
    payload: Command98_ReadCommandActionResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        action_number: 0 (UBInt8)
        command: 0 (UBInt16)
        byte_count: 0 (UBInt8)
        data: Array():
```

```
        [
        ]
    check_byte: 98 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14
\x86\x80\x00\x00\x00\x00\x62\x06\x00\x00\x00\x00\x00\x00\x62
```

Command 99

Command 99 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(99)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 99 (Command)
        byte_count: 4 (UBInt8)
        payload: Command99_ConfigureCommandActionRequest():
            action_number: 0 (UBInt8)
            command: 0 (UBInt16)
            byte_count: 0 (UBInt8)
            data: Array():
            [
            ]
        check_byte: 101 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12
\x82\x80\x00\x00\x00\x00\x63\x04\x00\x00\x00\x00\x65
# response
hf = HartFrame(99)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
```

```
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 99 (Command)
    byte_count: 6 (UBInt8)
    payload: Command99_ConfigureCommandActionResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        action_number: 0 (UBInt8)
        command: 0 (UBInt16)
        byte_count: 0 (UBInt8)
        data: Array():
        [
        ]
    check_byte: 99 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14
\x86\x80\x00\x00\x00\x00\x63\x06\x00\x00\x00\x00\x00\x00\x63
```

Command 100

Command 100 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(100)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 100 (Command)
        byte_count: 1 (UBInt8)
        payload: Command100_WritePrimaryVariableAlarmCodeRequest():
            primary_variable_alarm_selection_code: 0 (UBInt8)
        check_byte: 103 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x82\x80\x00\x00\x00\x00\x64\x01\x00\x67
# response
```

```
hf = HartFrame(100)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 100 (Command)
    byte_count: 3 (UBInt8)
    payload: Command100_WritePrimaryVariableAlarmCodeResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        primary_variable_alarm_selection_code: 0 (UBInt8)
    check_byte: 97 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11
\x86\x80\x00\x00\x00\x00\x64\x03\x00\x00\x00\x61
```

Command 101

Command 101 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(101)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 101 (Command)
        byte_count: 1 (UBInt8)
        payload: Command101_ReadSubDeviceToBurstMessageMapRequest():
            burst_message: 0 (UBInt8)
        check_byte: 102 (CheckByte)
```

```
encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x82\x80\x00\x00\x00\x00\x65\x01\x00\x66
# response
hf = HartFrame(101)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 101 (Command)
    byte_count: 5 (UBInt8)
    payload: Command101_ReadSubDeviceToBurstMessageMapResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        burst_message: 0 (UBInt8)
        subdevice_index: 0 (UBInt16)
    check_byte: 102 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13
\x86\x80\x00\x00\x00\x00\x65\x05\x00\x00\x00\x00\x00\x66
```

Command 102

Command 102 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(102)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
```

```
        command: 102 (Command)
        byte_count: 3 (UBInt8)
        payload: Command102_MapSubdeviceToBurstMessageRequest():
            burst_message: 0 (UBInt8)
            subdevice_index: 0 (UBInt16)
        check_byte: 103 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11
\x82\x80\x00\x00\x00\x00\x66\x03\x00\x00\x00\x67
# response
hf = HartFrame(102)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 102 (Command)
    byte_count: 5 (UBInt8)
    payload: Command102_MapSubdeviceToBurstMessageResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        burst_message: 0 (UBInt8)
        subdevice_index: 0 (UBInt16)
    check_byte: 101 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13
\x86\x80\x00\x00\x00\x00\x66\x05\x00\x00\x00\x00\x00\x65
```

Command 103

Command 103 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(103)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
```

```
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 103 (Command)
        byte_count: 9 (UBInt8)
        payload: Command103_WriteBurstPeriodRequest():
            burst_message: 0 (UBInt8)
            update_period: 00:00:00 (Time)
            maximum_update_period: 00:00:00 (Time)
        check_byte: 108 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17
\x82\x80\x00\x00\x00\x00\x67\x09\x00\x00\x00\x00\x00\x00\x00\x00\x00\x6c
# response
hf = HartFrame(103)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 103 (Command)
    byte_count: 11 (UBInt8)
    payload: Command103_WriteBurstPeriodResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        burst_message: 0 (UBInt8)
        update_period: 00:00:00 (Time)
        maximum_update_period: 00:00:00 (Time)
    check_byte: 106 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19
\x86\x80\x00\x00\x00\x00\x67\x0b\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x6a
```

Command 104

Command 104 example:

```
import oser
from htf.hart import HartFrame
```

```
# request
hf = HartFrame(104)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 104 (Command)
        byte_count: 8 (UBInt8)
        payload: Command104_WriteBurstTriggerRequest():
            burst_message: 0 (UBInt8)
            burst_trigger_mode_selection_code: 0 (UBInt8)
            device_variable_classification: 0 (UBInt8)
            units_code: 0 (UBInt8)
            trigger_level: 0.0 (BFloat)
        check_byte: 98 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16
\x82\x80\x00\x00\x00\x00\x68\x08\x00\x00\x00\x00\x00\x00\x00\x00\x62
# response
hf = HartFrame(104)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 104 (Command)
    byte_count: 10 (UBInt8)
    payload: Command104_WriteBurstTriggerResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        burst_message: 0 (UBInt8)
        burst_trigger_mode_selection_code: 0 (UBInt8)
        device_variable_classification: 0 (UBInt8)
        units_code: 0 (UBInt8)
        trigger_level: 0.0 (BFloat)
    check_byte: 100 (CheckByte)

encoded = hf.encode()
```

```
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18
\x86\x80\x00\x00\x00\x00\x68\x0a\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x64
```

Command 105

Command 105 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(105)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 105 (Command)
        byte_count: 1 (UBInt8)
        payload: Command105_ReadBurstModeConfigurationRequest():
            burst_message: 0 (UBInt8)
        check_byte: 106 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x82\x80\x00\x00\x00\x00\x69\x01\x00\x6a
# response
hf = HartFrame(105)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 105 (Command)
    byte_count: 31 (UBInt8)
    payload: Command105_ReadBurstModeConfigurationResponse():
        response_code: 'success' (UBInt8)
```

```
        field_device_status:  (FieldDeviceStatus)
        burst_mode_control_code: 0 (UBInt8)
        command_number_expansion_flag: 0 (UBInt8)
        device_variable_code_slot0: 0 (UBInt8)
        device_variable_code_slot1: 0 (UBInt8)
        device_variable_code_slot2: 0 (UBInt8)
        device_variable_code_slot3: 0 (UBInt8)
        device_variable_code_slot4: 0 (UBInt8)
        device_variable_code_slot5: 0 (UBInt8)
        device_variable_code_slot6: 0 (UBInt8)
        device_variable_code_slot7: 0 (UBInt8)
        burst_message: 0 (UBInt8)
        maximum_burst_messages: 0 (UBInt8)
        extended_command_number: 0 (UBInt16)
        update_time: 00:00:00 (Time)
        maximum_update_time: 00:00:00 (Time)
        burst_trigger_mode_code: 0 (UBInt8)
        device_variable_classification: 0 (UBInt8)
        units_code: 0 (UBInt8)
        trigger_value: 0.0 (BFloat)
    check_byte: 112 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25| 26| 27| 28| 29| 30| 31| 32| 33| 34| 35| 36| 37| 38| 39
\x86\x80\x00\x00\x00\x00\x69\x1f\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
```

Command 106

Command 106 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(106)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 106 (Command)
        byte_count: 0 (UBInt8)
        payload: Command106_FlushDelayedResponsesRequest():
        check_byte: 104 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
```

```
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x6a\x00\x68
# response
hf = HartFrame(106)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 106 (Command)
    byte_count: 2 (UBInt8)
    payload: Command106_FlushDelayedResponsesResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
    check_byte: 110 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10
\x86\x80\x00\x00\x00\x00\x6a\x02\x00\x00\x6e
```

Command 107

Command 107 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(107)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 107 (Command)
        byte_count: 9 (UBInt8)
        payload: Command107_WriteBurstDeviceVariablesRequest():
            device_variable_code_slot0: 0 (UBInt8)
```

```
            device_variable_code_slot1: 0 (UBInt8)
            device_variable_code_slot2: 0 (UBInt8)
            device_variable_code_slot3: 0 (UBInt8)
            device_variable_code_slot4: 0 (UBInt8)
            device_variable_code_slot5: 0 (UBInt8)
            device_variable_code_slot6: 0 (UBInt8)
            device_variable_code_slot7: 0 (UBInt8)
            burst_message: 0 (UBInt8)
        check_byte: 96 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17
\x82\x80\x00\x00\x00\x00\x6b\x09\x00\x00\x00\x00\x00\x00\x00\x00\x00\x60
# response
hf = HartFrame(107)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 107 (Command)
    byte_count: 11 (UBInt8)
    payload: Command107_WriteBurstDeviceVariablesResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        device_variable_code_slot0: 0 (UBInt8)
        device_variable_code_slot1: 0 (UBInt8)
        device_variable_code_slot2: 0 (UBInt8)
        device_variable_code_slot3: 0 (UBInt8)
        device_variable_code_slot4: 0 (UBInt8)
        device_variable_code_slot5: 0 (UBInt8)
        device_variable_code_slot6: 0 (UBInt8)
        device_variable_code_slot7: 0 (UBInt8)
        burst_message: 0 (UBInt8)
    check_byte: 102 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19
\x86\x80\x00\x00\x00\x00\x6b\x0b\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x66
```

Command 108

Command 108 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(108)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 108 (Command)
        byte_count: 3 (UBInt8)
        payload: Command108_WriteBurstModeCommandNumberRequest():
            command: 0 (UBInt16)
            burst_message: 0 (UBInt8)
        check_byte: 109 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11
\x82\x80\x00\x00\x00\x00\x6c\x03\x00\x00\x00\x6d
# response
hf = HartFrame(108)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 108 (Command)
    byte_count: 5 (UBInt8)
    payload: Command108_WriteBurstModeCommandNumberResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        command: 0 (UBInt16)
        burst_message: 0 (UBInt8)
    check_byte: 111 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13
\x86\x80\x00\x00\x00\x00\x6c\x05\x00\x00\x00\x00\x00\x6f
```

Command 109

Command 109 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(109)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 109 (Command)
        byte_count: 2 (UBInt8)
        payload: Command109_BurstModeControlRequest():
            burst_mode_control_code: 0 (UBInt8)
            burst_message: 0 (UBInt8)
        check_byte: 109 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10
\x82\x80\x00\x00\x00\x00\x6d\x02\x00\x00\x6d
# response
hf = HartFrame(109)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 109 (Command)
    byte_count: 4 (UBInt8)
    payload: Command109_BurstModeControlResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        burst_mode_control_code: 0 (UBInt8)
        burst_message: 0 (UBInt8)
    check_byte: 111 (CheckByte)
```

```
encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12
\x86\x80\x00\x00\x00\x00\x6d\x04\x00\x00\x00\x00\x6f
```

Command 110

Command 110 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(110)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 110 (Command)
        byte_count: 0 (UBInt8)
        payload: Command110_ReadAllDynamicVariablesRequest():
        check_byte: 108 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x82\x80\x00\x00\x00\x00\x6e\x00\x6c
# response
hf = HartFrame(110)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 110 (Command)
    byte_count: 2 (UBInt8)
    payload: Command110_ReadAllDynamicVariablesResponse():
```

```
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        dynamic_variables: Array():
        [
        ]
    check_byte: 106 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10
\x86\x80\x00\x00\x00\x00\x6e\x02\x00\x00\x6a
```

Command 111

Command 111 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(111)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 111 (Command)
        byte_count: 5 (UBInt8)
        payload: Command111_TransferServiceControlRequest():
            port: 0 (UBInt8)
            function_code: 0 (UBInt8)
            maximum_segment_length: 0 (UBInt8)
            starting_master_byte_counter: 0 (UBInt16)
        check_byte: 104 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13
\x82\x80\x00\x00\x00\x00\x6f\x05\x00\x00\x00\x00\x00\x68
# response
hf = HartFrame(111)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
```

```
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 111 (Command)
    byte_count: 9 (UBInt8)
    payload: Command111_TransferServiceControlResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        port: 0 (UBInt8)
        function_code: 0 (UBInt8)
        maximum_segment_length: 0 (UBInt8)
        starting_master_byte_counter: 0 (UBInt16)
        starting_field_device_byte_counter: 0 (UBInt16)
    check_byte: 96 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17
\x86\x80\x00\x00\x00\x00\x6f\x09\x00\x00\x00\x00\x00\x00\x00\x00\x00\x60
```

Command 112

Command 112 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(112)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 112 (Command)
        byte_count: 7 (UBInt8)
        payload: Command112_BlockTransferRequest():
            port: 0 (UBInt8)
            function_code: 0 (UBInt8)
            number_of_bytes_transferred: 0 (UBInt8)
            master_byte_count: 0 (UBInt16)
            slave_byte_count: 0 (UBInt16)
            data: Array():
            [
            ]
```

```
        check_byte: 117 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15
\x82\x80\x00\x00\x00\x00\x70\x07\x00\x00\x00\x00\x00\x00\x00\x75
# response
hf = HartFrame(112)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 112 (Command)
    byte_count: 9 (UBInt8)
    payload: Command112_BlockTransferResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        port: 0 (UBInt8)
        function_code: 0 (UBInt8)
        number_of_bytes_transferred: 0 (UBInt8)
        master_byte_count: 0 (UBInt16)
        slave_byte_count: 0 (UBInt16)
        data: Array():
        [
        ]
    check_byte: 127 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17
\x86\x80\x00\x00\x00\x00\x70\x09\x00\x00\x00\x00\x00\x00\x00\x00\x00\x7f
```

Command 113

Command 113 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(113)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
```

```
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 113 (Command)
        byte_count: 15 (UBInt8)
        payload: Command113_CatchDeviceVariableRequest():
            destination_device_variable: 0 (UBInt8)
            capture_mode_code: 0 (UBInt8)
            source_slave_expanded_device_type_code: 0 (UBInt16)
            source_slave_device_id: BitStruct():
                value: 0 (BitField(24))
            command_number_expansion_flag: 0 (UBInt8)
            source_slot_number: 0 (UBInt8)
            shed_time: 0.0 (BFloat)
            source_command_number: 0 (UBInt16)
        check_byte: 124 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23
\x82\x80\x00\x00\x00\x00\x71\x0f\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x7c
# response
hf = HartFrame(113)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 113 (Command)
    byte_count: 17 (UBInt8)
    payload: Command113_CatchDeviceVariableResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        destination_device_variable: 0 (UBInt8)
        capture_mode_code: 0 (UBInt8)
        source_slave_address: BitStruct():
            value: 0 (BitField(40))
        command_number_expansion_flag: 0 (UBInt8)
        source_slot_number: 0 (UBInt8)
        shed_time: 0.0 (BFloat)
        source_command_number: 0 (UBInt16)
    check_byte: 102 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
```

```
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25
\x86\x80\x00\x00\x00\x00\x71\x11\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x6
```

Command 114

Command 114 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(114)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 114 (Command)
        byte_count: 1 (UBInt8)
        payload: Command114_ReadCaughtDeviceVariableRequest():
            destination_device_variable: 0 (UBInt8)
        check_byte: 113 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x82\x80\x00\x00\x00\x00\x72\x01\x00\x71
# response
hf = HartFrame(114)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 114 (Command)
    byte_count: 17 (UBInt8)
    payload: Command114_ReadCaughtDeviceVariableResponse():
        response_code: 'success' (UBInt8)
```

```
        field_device_status:  (FieldDeviceStatus)
        destination_device_variable: 0 (UBInt8)
        capture_mode_code: 0 (UBInt8)
        source_slave_address: BitStruct():
            value: 0 (BitField(40))
        LSB_of_source_command_number: 0 (UBInt8)
        source_slot_number: 0 (UBInt8)
        shed_time: 0.0 (BFloat)
        source_command_number: 0 (UBInt16)
    check_byte: 101 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25
\x86\x80\x00\x00\x00\x00\x72\x11\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x6
```

Command 115

Command 115 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(115)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 115 (Command)
        byte_count: 1 (UBInt8)
        payload: Command115_ReadEventNotificationSummaryRequest():
            event_specification_number: 0 (UBInt8)
        check_byte: 112 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x82\x80\x00\x00\x00\x00\x73\x01\x00\x70
# response
hf = HartFrame(115)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
```

```
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 115 (Command)
    byte_count: 22 (UBInt8)
    payload: Command115_ReadEventNotificationSummaryResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        event_specification_number: 0 (UBInt8)
        number_of_events_supported: 0 (UBInt8)
        event_status_notification_control: BitStruct():
            event_status: 0 (Nibble)
            notification_control_code: 0 (Nibble)
        first_event_trigger_time: 00:00:00 (Time)
        event_notification_retry_time: 00:00:00 (Time)
        maximum_update_time: 00:00:00 (Time)
        event_debounce_interval: 00:00:00 (Time)
        device_status_mask: BitStruct():
            value: 0 (BitField(8))
        event_mask: Array():
        [
        ]
    check_byte: 99 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25| 26| 27| 28| 29| 30
\x86\x80\x00\x00\x00\x00\x73\x16\x00\x00\x00\x00
```

```
        ]
        command: 116 (Command)
        byte_count: 2 (UBInt8)
        payload: Command116_WriteEventNotificationBitMaskRequest():
            event_specification_number: 0 (UBInt8)
            device_status_mask: BitStruct():
                value: 0 (BitField(8))
            event_mask: Array():
            [
            ]
        check_byte: 116 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10
\x82\x80\x00\x00\x00\x00\x74\x02\x00\x00\x74
# response
hf = HartFrame(116)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 116 (Command)
    byte_count: 4 (UBInt8)
    payload: Command116_WriteEventNotificationBitMaskResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        event_specification_number: 0 (UBInt8)
        device_status_mask: BitStruct():
            value: 0 (BitField(8))
        event_mask: Array():
        [
        ]
    check_byte: 118 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12
\x86\x80\x00\x00\x00\x00\x74\x04\x00\x00\x00\x00\x76
```

Command 117

Command 117 example:

```
import oser
from htf.hart import HartFrame
```

```
# request
hf = HartFrame(117)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 117 (Command)
        byte_count: 13 (UBInt8)
        payload: Command117_WriteEventNotificationTimingRequest():
            event_specification_number: 0 (UBInt8)
            event_notification_retry_time: 00:00:00 (Time)
            maximum_update_time: 00:00:00 (Time)
            event_debounce_interval: 00:00:00 (Time)
        check_byte: 122 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21
\x82\x80\x00\x00\x00\x00\x75\x0d\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x7a
# response
hf = HartFrame(117)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 117 (Command)
    byte_count: 15 (UBInt8)
    payload: Command117_WriteEventNotificationTimingResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        event_specification_number: 0 (UBInt8)
        event_notification_retry_time: 00:00:00 (Time)
        maximum_update_time: 00:00:00 (Time)
        event_debounce_interval: 00:00:00 (Time)
    check_byte: 124 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
```

```
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23
\x86\x80\x00\x00\x00\x00\x75\x0f\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x7c
```

Command 118

Command 118 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(118)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 118 (Command)
        byte_count: 2 (UBInt8)
        payload: Command118_EventNotificationControlRequest():
            event_specification_number: 0 (UBInt8)
            event_notification_control_code: 0 (UBInt8)
        check_byte: 118 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10
\x82\x80\x00\x00\x00\x00\x76\x02\x00\x00\x76
# response
hf = HartFrame(118)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 118 (Command)
    byte_count: 4 (UBInt8)
    payload: Command118_EventNotificationControlResponse():
        response_code: 'success' (UBInt8)
```

```
        field_device_status:  (FieldDeviceStatus)
        event_specification_number: 0 (UBInt8)
        event_notification_control_code: 0 (UBInt8)
    check_byte: 116 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12
\x86\x80\x00\x00\x00\x00\x76\x04\x00\x00\x00\x00\x74
```

Command 119

Command 119 example:

```
import oser
from htf.hart import HartFrame
# request
hf = HartFrame(119)
print(hf)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: False (Flag)
            address: 0x0 (SlaveAddress(38))
        expansion_bytes: Array():
        [
        ]
        command: 119 (Command)
        byte_count: 8 (UBInt8)
        payload: Command119_AcknowledgeEventNotificationRequest():
            event_specification_number: 0 (UBInt8)
            first_event_trigger_time: 00:00:00 (Time)
            configuration_change_counter: 0 (UBInt16)
            device_status: BitStruct():
                value: 0 (BitField(8))
            command48_data: Array():
            [
            ]
        check_byte: 125 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16
\x82\x80\x00\x00\x00\x00\x77\x08\x00\x00\x00\x00\x00\x00\x00\x00\x7d
# response
hf = HartFrame(119)
hf.delimiter.frame_type.set("ack")
print(hf)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
```

```
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: False (Flag)
        address: 0x0 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 119 (Command)
    byte_count: 10 (UBInt8)
    payload: Command119_AcknowledgeEventNotificationResponse():
        response_code: 'success' (UBInt8)
        field_device_status:  (FieldDeviceStatus)
        event_specification_number: 0 (UBInt8)
        first_event_trigger_time: 00:00:00 (Time)
        configuration_change_counter: 0 (UBInt16)
        device_status: BitStruct():
            value: 0 (BitField(8))
        command48_data: Array():
        [
        ]
    check_byte: 123 (CheckByte)

encoded = hf.encode()
print(oser.to_hex(encoded))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18
\x86\x80\x00\x00\x00\x00\x77\x0a\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x7b
```

Device Specific Commands

To realize device specific commands `HartFrame` must be subclassed and the methods `requests_generator` and `responses_generator` must be overwritten.

Both methods are generators that yield tuples consisting of the command number and a device specific parser.

It is recommended to use `oser.Lazy` to wrap the parser to speed up the initialization of the HART frame.

In the following example the command 48 and 181 are implemented:

```
from oser import \
    ByteStruct, BitStruct, BitField, Switch, String, \
    Enum, UBInt8, Lazy, BFloat, to_hex
from htf.hart import HartFrame, ResponseCode, FieldDeviceStatus


class Command48_ReadAdditionalDeviceStatusRequest(ByteStruct):
    def __init__(self):
        super(Command48_ReadAdditionalDeviceStatusRequest, self).__init__()
        self.device_specific_status = BitStruct()
        self.device_specific_status.value = BitField(length=48)
        self.extended_device_status = UBInt8()
        self.device_operating_mode = UBInt8()
        self.status0 = UBInt8()
```

```
        self.status1 = UBInt8()
        self.analog_channel_saturated = UBInt8()
        self.status2 = UBInt8()
        self.status3 = UBInt8()
        self.analog_channel_fixed = UBInt8()
        self.device_specific_status = BitStruct()
        self.device_specific_status.value = BitField(length=88)


class Command48_ReadAdditionalDeviceStatusResponse(ByteStruct):
    def __init__(self):
        super(Command48_ReadAdditionalDeviceStatusResponse, self).__init__()

        self.response_code = ResponseCode(
                error_device_specific_command_error=6,
                warning_update_in_progress=8,
                error_access_restricted=16,
                error_busy=32)
        self.field_device_status = FieldDeviceStatus()

        self.device_specific_status = BitStruct()
        self.device_specific_status.value = BitField(length=48)
        self.extended_device_status = UBInt8()
        self.device_operating_mode = UBInt8()
        self.status0 = UBInt8()
        self.status1 = UBInt8()
        self.analog_channel_saturated = UBInt8()
        self.status2 = UBInt8()
        self.status3 = UBInt8()
        self.analog_channel_fixed = UBInt8()
        self.device_specific_status = BitStruct()
        self.device_specific_status.value = BitField(length=88)


class Command181_SetAnalogOutputRequest(ByteStruct):
    def __init__(self):
        super(Command181_SetAnalogOutputRequest, self).__init__()
        self.temporary_analog_output = Enum(
            prototype=UBInt8,
            values={
                    "invalid": 0x07,
                    "beamblock_signal": 0x06,
                    "fixed_current": 0x05,
                    "current_corresponding_to_gas_concentration": 0x04,
                    "maintenance_signal": 0x03,
                    "error_signal": 0x02,
                    "warning_signal": 0x01,
                    "analog_normal": 0x00,
                    },
            value="invalid",
            strict=True)
        self.gas_concentration = BFloat(.0)
        self.fixed_current_mA = BFloat(.0)


class Command181_SetAnalogOutputResponse(ByteStruct):
    def __init__(self):
        super(Command181_SetAnalogOutputResponse, self).__init__()

        self.response_code = ResponseCode()
```

```
        self.field_device_status = FieldDeviceStatus()

        self.temporary_analog_output = Enum(
            prototype=UBInt8,
            values={
                    "invalid": 0x07,
                    "beamblock_signal": 0x06,
                    "fixed_current": 0x05,
                    "current_corresponding_to_gas_concentration": 0x04,
                    "maintenance_signal": 0x03,
                    "error_signal": 0x02,
                    "warning_signal": 0x01,
                    "analog_normal": 0x00,
                    },
            value="invalid",
            strict=True)
        self.gas_concentration = BFloat(.0)
        self.fixed_current_mA = BFloat(.0)


class DeviceHartFrame(HartFrame):

    def requests_generator(self):
        # this is only needed in subclass hierarchies
        for command, parser in super(DeviceHartFrame, self).requests_generator():
            yield command, parser
        # the following commands are device specific
        yield 48, Lazy(Command48_ReadAdditionalDeviceStatusRequest)
        yield 181, Lazy(Command181_SetAnalogOutputRequest)

    def responses_generator(self):
        # this is only needed in subclass hierarchies
        for command, parser in super(DeviceHartFrame, self).responses_generator():
            yield command, parser
        # the following commands are device specific
        yield 48, Lazy(Command48_ReadAdditionalDeviceStatusResponse)
        yield 181, Lazy(Command181_SetAnalogOutputResponse)
```

These commands can be easily adapted for the customer's needs.

In the following example the usage is demonstrated.

```
# command 48
hf = DeviceHartFrame(command=48)

# request
hf.delimiter.frame_type.set("stx")

binary = hf.encode()
print(hf)
    DeviceHartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
```

```
            master_address: 'primary_master' (Flag)
            burst_mode: 0 (Flag)
            address: 0 (BitField(38))
        expansion_bytes: Array():
        [
        ]
        command: 48 (UBInt8)
        byte_count: 19 (UBInt8)
        payload: Command48_ReadAdditionalDeviceStatusRequest():
            device_specific_status: BitStruct():
                value: 0 (BitField(88))
            extended_device_status: 0 (UBInt8)
            device_operating_mode: 0 (UBInt8)
            status0: 0 (UBInt8)
            status1: 0 (UBInt8)
            analog_channel_saturated: 0 (UBInt8)
            status2: 0 (UBInt8)
            status3: 0 (UBInt8)
            analog_channel_fixed: 0 (UBInt8)
        check_byte: 33 (CheckByte)

print(to_hex(binary))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25| 26| 27
\x82\x80\x00\x00\x00\x00\x30\x13\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0

# response
hf.delimiter.frame_type.set("ack")
binary = hf.encode()
print(hf)
    DeviceHartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'ack' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: 0 (Flag)
            address: 0 (BitField(38))
        expansion_bytes: Array():
        [
        ]
        command: 48 (UBInt8)
        byte_count: 0 (UBInt8)
        payload: Command48_ReadAdditionalDeviceStatusResponse():
        check_byte: 54 (CheckByte)

print(to_hex(binary))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x86\x80\x00\x00\x00\x00\x30\x00\x36

# command 181
hf = DeviceHartFrame(command=181)
print(hf)
    DeviceHartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
```

```
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: 0 (Flag)
            address: 0 (BitField(38))
        expansion_bytes: Array():
        [
        ]
        command: 181 (UBInt8)
        byte_count: 0 (UBInt8)
        payload: Command181_SetAnalogOutputRequest():
            temporary_analog_output: 'invalid' (UBInt8)
            gas_concentration: 0.0 (BFloat)
            fixed_current_mA: 0.0 (BFloat)
        check_byte: 0 (CheckByte)

# request
hf.delimiter.frame_type.set("stx")
hf.payload.temporary_analog_output.set(0x04)
hf.payload.gas_concentration.set(15.0)
hf.payload.fixed_current_mA.set(4.5)

binary = hf.encode()
print(hf)
    DevieHartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: 0 (Flag)
            address: 0 (BitField(38))
        expansion_bytes: Array():
        [
        ]
        command: 181 (UBInt8)
        byte_count: 9 (UBInt8)
        payload: Command181_SetAnalogOutputRequest():
            temporary_analog_output: 'current_corresponding_to_gas_concentration' (UBInt8)
            gas_concentration: 15.0 (BFloat)
            fixed_current_mA: 4.5 (BFloat)
        check_byte: 91 (CheckByte)

print(to_hex(binary))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17
\x82\x80\x00\x00\x00\x00\xb5\x09\x04\x41\x70\x00\x00\x40\x90\x00\x00\x5b

# response
hf.delimiter.frame_type.set("ack")
binary = hf.encode()
print(hf)
    DeviceHartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'ack' (BitField(3))
        address: Address():
```

```
            master_address: 'primary_master' (Flag)
            burst_mode: 0 (Flag)
            address: 0 (BitField(38))
        expansion_bytes: Array():
        [
        ]
        command: 181 (UBInt8)
        byte_count: 11 (UBInt8)
        payload: Command181_SetAnalogOutputResponse():
            response_code: 'success' (UBInt8)
            field_device_status:  (FieldDeviceStatus)
            temporary_analog_output: 'invalid' (UBInt8)
            gas_concentration: 0.0 (BFloat)
            fixed_current_mA: 0.0 (BFloat)
        check_byte: 191 (CheckByte)

print(to_hex(binary))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19
\x86\x80\x00\x00\x00\x00\xb5\x0b\x00\x00\x07\x00\x00\x00\x00\x00\x00\x00\x00\xbf
```

Protocol Factory

A protocol factory creates device specific HART frame abstractions respecting the device type or device address.

It is useful for the HART-shell and the HART-monitor.

A protocol factory needs two methods that return a device specific HART frame by device type or device address (in which the device type is included).

```
from htf.hart import HartFrame
from htf.hart.universal_commands import \
    Command48_ReadAdditionalDeviceStatusRequest,\
    Command48_ReadAdditionalDeviceStatusResponse
from oser import Lazy
import logging
logger = logging.getLogger(__name__)
logging.basicConfig(level=logging.DEBUG)


class DeviceHartFrame(HartFrame):
    def requests_generator(self):
        # here come the device specific commands
        yield 48, Lazy(Command48_ReadAdditionalDeviceStatusRequest)

    def responses_generator(self):
        # here come the device specific commands
        yield 48, Lazy(Command48_ReadAdditionalDeviceStatusResponse)

_frame_parser_db = {
    # the two LSBs in the device type cannot be used in the
    # device address so they must be masked
    0x1337 & 0x3fff: ["SpecificHartDevice", DeviceHartFrame],
    # ...
}
```

```
def get_hart_frame_by_device_type(device_type):
    """
    Create a HartFrame with device specific commands
    for the given device type.

    Args:
        device_type (int): the device type to be used

    Returns:
        HartFrame: a HartFrame generator for device specific commands
    """
    logger.info("Selecting frame by device type 0x%010x" % device_type)
    # the two LSBs in the device type cannot be used in the
    # device address so they must be masked
    device_type = device_type & 0x3fff
    try:
        name, DeviceSpecificHartFrame = _frame_parser_db[device_type]
        logger.info(
            "Found specific frame parser for %s (0x%x)" %
            (name, device_type))

        def _generator(command=None, auto_length=True):
            return DeviceSpecificHartFrame(command=command, auto_length=auto_length)

        return _generator
    except KeyError:
        logger.info("No specific frame parser found for device id 0x%010x. "
                    "Using default HartFrame()." % device_type)
        return HartFrame


def get_hart_frame_by_device_address(device_address):
    """
    Create a HartFrame with device specific commands
    for the given device address by extracting the
    device type from the device address

    Args:
        device_address (int): the device address to be used

    Returns:
        HartFrame: a HartFrame generator for device specific commands
    """
    device_type = device_address >> 24
    return get_hart_frame_by_device_type(device_type)
```

An exemplary usage looks like:

```
DeviceSpecificHartFrame = get_hart_frame_by_device_type(device_type=0x1337)
hf = DeviceSpecificHartFrame(command=48)
print(hf)
    DeviceHartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
```

```
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: 0 (Flag)
            address: 0 (BitField(38))
        expansion_bytes: Array():
        [
        ]
        command: 48 (UBInt8)
        byte_count: 0 (UBInt8)
        payload: Command48_ReadAdditionalDeviceStatusRequest():
            device_specific_status: BitStruct():
                value: 0 (BitField(88))
            extended_device_status: 0 (UBInt8)
            device_operating_mode: 0 (UBInt8)
            status0: 0 (UBInt8)
            status1: 0 (UBInt8)
            analog_channel_saturated: 0 (UBInt8)
            status2: 0 (UBInt8)
            status3: 0 (UBInt8)
            analog_channel_fixed: 0 (UBInt8)
        check_byte: 0 (CheckByte)

hf = DeviceSpecificHartFrame(command=181)
print(hf)
    DeviceHartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: 0 (Flag)
            address: 0 (BitField(38))
        expansion_bytes: Array():
        [
        ]
        command: 181 (UBInt8)
        byte_count: 0 (UBInt8)
        payload: b''
        check_byte: 0 (CheckByte)


DeviceSpecificHartFrame = get_hart_frame_by_device_address(device_address=0x1337000023)
INFO:__main__:Selecting frame by device type 0x0000001337
INFO:__main__:Found specific frame parser for SpecificHartDevice (0x1337)
hf = DeviceSpecificHartFrame(command=48)
print(hf)
    DeviceHartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'stx' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: 0 (Flag)
            address: 0 (BitField(38))
        expansion_bytes: Array():
```

```
        [
        ]
        command: 48 (UBInt8)
        byte_count: 0 (UBInt8)
        payload: Command48_ReadAdditionalDeviceStatusRequest():
            device_specific_status: BitStruct():
                value: 0 (BitField(88))
            extended_device_status: 0 (UBInt8)
            device_operating_mode: 0 (UBInt8)
            status0: 0 (UBInt8)
            status1: 0 (UBInt8)
            analog_channel_saturated: 0 (UBInt8)
            status2: 0 (UBInt8)
            status3: 0 (UBInt8)
            analog_channel_fixed: 0 (UBInt8)
        check_byte: 0 (CheckByte)
```

HART-Shell

The HART-shell enables you to develop and debug a HART device interactively. It is intended to be used interactively but can also be used non-interactively. The HART-shell is also customizable.

```
class htf.hart.HartShell(comport: str | None = None, decoder_generator: Type[HartFrame] |
Callable[[...], HartFrame] | None = None, protocol_factory: Callable[[...], Type[HartFrame]]
| None = None, timeout: int | float | None = None, interactive: bool = True, loglevel: int |
None = 2, argv: List[str] | None = None)
```

A HART Shell that can be used interactively.

Parameters:

- **comport** – the comport to be used where the HART modem is connected to.
- **decoder_generator=None** – a callable that returns a HartFrame instance with or without device specific commands.
- **protocol_factory=None** – a protocol factory that returns a device specific HART frame parser with respect to the supplied device type.
- **timeout=None** – the default timeout in seconds.
- **interactive=True** – if set to `True` the HART Shell becomes interactive.
- **loglevel=2** – set loglevel for loggnig to `loglevel`.
- **argv=None** – used for testing, etc.

```
close() → None
```

Close the HART application layer and close the HART interface supplied. Is called in __del__, too.

```
cmd(command: int, inputs: List[Any] | None = None, edit_all: bool = False, interactive:
bool = True, timeout: int | float | None = None) → HartFrame | None
```

Query a HART command.

Parameters:

- **command** – the HART command number.
- **inputs=None** – the inputs to be set in order to be put into the HART request.
- **edit_all=False** – if set to `True` and the HART shell runs interactively all request fields are edited interactively.
- **interactive=True** – if set to `True` the request is edited interactively.
- **timeout=None** – the timeout in seconds. If `None` is supplied the default timeout is used. If the default timeout is not set 1.0 is used.

```
find_device(polling_addresses: List[int] | None = None) → int | None
```

Find the first HART device by polling all addresses stated in `polling_addresses` and use the first device found.

If no device is found an `AssertionError` is raised.

Parameters:
polling_addresses=[0] – the polling addresses used to find a HART device.
Returns:
Unique HART address of the first device found in polling.
Return type:
int

```
last_preamble_count() → int
```

Returns the number of preambles most recently received by the HartInterface.

Returns:
number of preambles
Return type:
int

```
last_response() → HartFrame | Any | None
```

Get the last response.

The last received packet can also be access via `self.response`.

Returns:
the last received packet.
Return type:
object

```
list_device_commands(commands: List[int] | None = None) → List[int]
```

List the implemented commands of a HART device. Helps to find undocumented commands.

Parameters:
commands – the list of commands to be probed. If set to `None` `0 .. 255` is used.
Returns:
the implemented commands.
Return type:
list of int

```
poll_devices(polling_addresses: List[int] | None = None) → None
```

Poll `polling_addresses` to find HART devices.

Parameters:
polling_addresses=[0] – list of HART polling addresses

```
set_decoder_by_device_address(device_address: int) → None
```

Select the frame decoder and encoder by the device address

Parameters:
device_address – the device address

```
set_decoder_generator(decoder_generator: Type[HartFrame] | Callable[[...], HartFrame] |
None = None) → None
```

Set the frame generator.

Parameters:
decoder_generator=HartFrame – a parser for HART frames.

```
set_device_address(device_address: int) → None
```

Set the current HART device address.

Parameters:
device_address – the HART device address to be used (<= 38 bits)

```
set_loglevel(loglevel: int | None) → None
```

Set the loglevel.

Loglevel `0` means `quiet`.

Loglevel `1` means `binary` output. All packets are printed in a compact binary format.

Loglevel `2` means `binary+payload of interest` output. All packets are printed in a compact binary format. Additionally the interesting payload is printed. `response_code` is left out if it is `success` and `field_device_status` is left out in case of `0`. Other payload is printed line by line.

Loglevel `3` means `binary+whole packet` output. All packets are printed in a compact binary format. Additionally the packets are fully printed.

Parameters:
loglevel – Set loglevel to `loglevel`.

```
set_master_address(master_address: int) → None
```

Set the current simulated master address.

Parameters:
master_address – “primary_master” or “secondary_master”

```
set_number_of_preambles(number_of_preambles: int) → None
```

Set the number of preambles for requests.

Parameters:
number_of_preambles – the number of preambles for requests.

```
set_protocol_factory(protocol_factory: Callable[[...], Type[HartFrame]] | None = None) →
None
```

Set the protocol factory.

Parameters:
protocol_factory=None – a protocol factory that returns a device specific HART frame parser with respect to the supplied device type.

```
set_timeout(timeout: int | float | None) → None
```

Set the default timeout.

Parameters:
timeout – the default timeout in seconds

Interactive usage

`htf` comes with a HART-shell command line utility called `hart-shell`.

When run interactively all methods in `htf.hart.HartShell` are put into the local namespace so they can be run by entering the name.

Additionally macros are created for each possible command, eg. `cmd0`, `cmd1`, `..`, `cmd255`. These can be run by name, too.

The command line utility is based on IPython. IPython allows tab-completion. To enter long commands enter the first characters and press `TAB` and select the command using the arrow keys.

To run the `hart-shell` simply enter

```
$ hart-shell -c <comport>
```

Show help

```
$ hart-shell -h
usage: hart-shell [-h] -c COMPORT [-d Protocol database] [-t timeout]
                  [-v loglevel] [-V]

hart-shell - HILSTER Testing Framework (HILSTER GmbH, http://hilster.io)

optional arguments:
  -h, --help            show this help message and exit
  -c COMPORT, --comport COMPORT
                        comport for HART communication
  -d Protocol database, --database Protocol database
                        An import string to the protocol database where device
                        specific HART frames can be found. See docs for
                        details.
  -t timeout, --timeout timeout
                        The default timeout for commands.
  -v loglevel, --loglevel loglevel
                        The loglevel: 0: quiet, 1: binary, 2: binary+payload
                        of interest, 3: binary+debug
  -V, --version         Show version and exit
```

Find and select the first device

To find and select a device `htf.hart.HartShell.find_device` is used. If a device or multiple devices are found the first device address is selected and the address is used for all other commands entered.

Find and select the first device for polling address `0`:

```
In [1]: find_device()
Using first device with address 0x1337000023
```

Find and select the first device for polling addresses `0..9`:

```
In [2]: find_device(range(10))
Using first device with address 0x1337000023
```

Poll devices

To poll HART devices `htf.hart.HartShell.poll_devices` is used.

To poll polling address `0` enter:

```
In [1]: poll_devices()
Found 1 HART device.
0x1337000023
```

To poll polling addresses `0..9` enter:

```
In [2]: poll_devices(range(10))
Found 1 HART device.
0x1337000023
```

Set the timeout

The timeout for commands can be set using `htf.hart.HartShell.set_timeout`.

To set the timeout to `3.0` seconds enter:

```
In [1]: set_timeout(3.0)
```

Device selection

To select a device manually the device address can be set using `htf.hart.HartShell.set_device_address`.

To set the device address to `0x1337000024` enter:

```
In [1]: set_device_address(0x1337000024)
```

Running commands interactively

You can run commands interactively to debug, test and develop a HART-device. The easiest way is to enter `cmd` plus the command number to be run, eg. `cmd1` to run command 1.

By default the device address selected by `htf.hart.HartShell.find_device` or set by `htf.hart.HartShell.set_device_address` is used automatically.

To select a device and run command 1 on it enter:

```
In [1]: find_device()
Using first device with address 0x1337000023
Out[1]: 82527125539

In [2]: cmd1()
Running Command 1
Request:
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'stx' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: 0 (Flag)
        address: 0 (BitField(38))
    expansion_bytes: Array():
    [
    ]
    command: 1 (UBInt8)
    byte_count: 0 (UBInt8)
    payload: Command1_ReadPrimaryVariableRequest():
    check_byte: 0 (CheckByte)

Response:
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: 0 (Flag)
        address: 82527125539 (BitField(38))
    expansion_bytes: Array():
    [
    ]
    command: 1 (UBInt8)
    byte_count: 7 (UBInt8)
    payload: Command1_ReadPrimaryVariableResponse():
        response_code: 'buffer_overflow' (UBInt8)
        field_device_status: non-primary_variable_out_of_limits (FieldDeviceStatus)
        units: 123 (UBInt8)
        primary_variable: 456.78900146484375 (BFloat)
    check_byte: 193 (CheckByte)

Out[2]: <htf.hart.protocol.HartFrame at 0x7fb4d4165080>
```

The timeout set by `htf.hart.HartShell.set_timeout` is used if not specified for the specific comamnd. To set the timeout add a `timeout=3.0` agrument.

```
In [3]: cmd1(timeout=3.0)
```

Loglevel

The loglevel can be set via commandline using the `-v` option or within the HART-Shell using

```
In [1]: set_loglevel(loglevel)
```

The values for `loglevel` reach from `0` to `3`. `0` means quiet. `1` means binary short form. `2` means binary short form and payload of interest. `3` means binary short form and debug.

The following example lists all loglevel usages:

```
In [1]: find_device()
Using first device with address 0x52eb021ce5
Out[1]: 356130102501

In [2]: set_loglevel(0)

In [3]: cmd1()
Out[3]: <htf.hart.protocol.HartFrame at 0x7419a90>

In [4]: set_loglevel(1)

In [5]: cmd1()
STX: 5FF 82 92EB021CE5 01
ACK: 5FF 86 92EB021CE5 00508B7FE00000 06
Out[5]: <htf.hart.protocol.HartFrame at 0x71da278>

In [6]: set_loglevel(2)

In [7]: cmd1()
STX: 5FF 82 92EB021CE5 01
ACK: 5FF 86 92EB021CE5 00508B7FE00000 06
field_device_status: 80
units: 139
primary_variable: nan
Out[7]: <htf.hart.protocol.HartFrame at 0x1f862e8>

In [8]: set_loglevel(3)

In [9]: cmd1()
STX: 5FF 82 92EB021CE5 01
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'stx' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: 0 (Flag)
        address: 0x52eb021ce5 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 1 (Command)
```

```
    byte_count: 0 (UBInt8)
    payload: Command1_ReadPrimaryVariableRequest():
    check_byte: 1 (CheckByte)

ACK: 5FF 86 92EB021CE5 00508B7FE00000 06
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: 0 (Flag)
        address: 0x12eb021ce5 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 1 (Command)
    byte_count: 7 (UBInt8)
    payload: Command1_ReadPrimaryVariableResponse():
        response_code: 'success' (UBInt8)
        field_device_status: configuration_changed; more_status_available (FieldDeviceStatus)
        units: 139 (UBInt8)
        primary_variable: nan (BFloat)
    check_byte: 6 (CheckByte)

Out[9]: <htf.hart.protocol.HartFrame at 0x6ed22b0>
```

The default loglevel is `2`.

Set the number of preambles

You can set the number of preambles for requests using

```
In [1]: set_number_of_preambles(3)

In [2]: cmd1()
STX: 3FF 82 92EB021CE5 01
ACK: 5FF 86 92EB021CE5 00508B7FE00000 06
field_device_status: 80
units: 139
primary_variable: 0.0
Out[2]: <htf.hart.protocol.HartFrame at 0x7238ef0>

In [3]: set_number_of_preambles(5)

In [4]: cmd1()
STX: 5FF 82 92EB021CE5 01
ACK: 5FF 86 92EB021CE5 00508B7FE00000 06
field_device_status: 80
units: 139
primary_variable: nan
Out[4]: <htf.hart.protocol.HartFrame at 0x6edab70>
```

Access last response

The last response can be accessed using `self.response`.

```
In [1]: find_device()
Using first device with address 0x52eb021ce5
Out[1]: 356130102501

In [2]: cmd1()
STX: 5FF 82 92EB021CE5 01
ACK: 5FF 86 92EB021CE5 00508B7FE00000 06
field_device_status: 80
units: 139
primary_variable: nan
Out[2]: <htf.hart.protocol.HartFrame at 0x7014240>

In [3]: print(self.response)
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: 0 (Flag)
        address: 0x12eb021ce5 (SlaveAddress(38))
    expansion_bytes: Array():
    [
    ]
    command: 1 (Command)
    byte_count: 7 (UBInt8)
    payload: Command1_ReadPrimaryVariableResponse():
        response_code: 'success' (UBInt8)
        field_device_status: configuration_changed; more_status_available (FieldDeviceStatus)
        units: 139 (UBInt8)
        primary_variable: 0.0 (BFloat)
    check_byte: 6 (CheckByte)
```

List device commands

You can poll a list of device commands to see which are supported. This way it is possible to find undocumented commands.

If you do not specify a list of commands to be polled `0 .. 255` is used.

```
In [1]: find_device()
Using first device with address 0x52eb021ce5
Out[1]: 356130102501

In [2]: list_device_commands()
command 0 .. is supported
command 1 .. is supported
command 2 .. is supported
command 3 .. is supported
```

```
command 4 .. is supported
command 5 .. is supported
command 6 .. is not supported
command 7 .. is not supported
command 8 .. is not supported
command 9 .. is not supported
command 10 .. is supported
command 11 .. is not supported
command 12 .. is supported
command 13 .. is not supported
command 14 .. is supported
command 15 .. is not supported
[...]
command 248 .. is supported
command 249 .. is supported
command 250 .. is supported
command 251 .. is supported
command 252 .. is supported
command 253 .. is supported
command 254 .. is supported
command 255 .. is supported
Out[2]:
[0,
 1,
 2,
 3,
 4,
 5,
 10,
 12,
 14,
 [...]
 248,
 249,
 250,
 251,
 252,
 253,
 254,
 255]
```

You can specify the commands to poll by adding `commands=<iterable>`:

```
In [2]: list_device_commands(commands=[1, 2, 3])
command 1 .. is supported
command 2 .. is supported
command 3 .. is supported
Out[2]: [1, 2, 3]
```

Master address

To set the master address `htf.hart.HartShell.set_master_address` is used.

To set the master address to *secondary_master* enter:

```
In [1]: set_master_address('secondary_master')
```

Exit

To exit `hart-shell` enter `exit()`, `exit` or press `CTRL+d` followed by `y` and `Return`.

Method documentation

You can easily see the documentation for a method by adding a `?` at the end of the command.

```
In [1]: set_master_address?
Signature: set_master_address(master_address)
Docstring:
Set the current simulated master address.

Args:
    master_address (str): "primary_master" or "secondary_master"
```

Editing command payload interactively

By default a commands payload can be changed interactively. If the payload is empty there is nothing to be changed and the command is sent immediately.

For example the command 6 (set polling address) has a changeable payload. To change the polling address to `1` enter:

```
In [1]: find_device()
Using first device with address 0x1337000023
Out[1]: 82527125539

In [2]: cmd6()
Running Command 6
Enter value for HartFrame.payload.polling_address (UBInt8) [0]> 1
Enter value for HartFrame.payload.loop_current_mode (UBInt8) [0]>
Request:
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'stx' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: 0 (Flag)
        address: 0 (BitField(38))
    expansion_bytes: Array():
    [
    ]
    command: 6 (UBInt8)
    byte_count: 0 (UBInt8)
    payload: Command6_WritePollingAddressRequest():
        polling_address: 1 (UBInt8)
```

```
        loop_current_mode: 0 (UBInt8)
    check_byte: 0 (CheckByte)
```

If you do not want to change a value simply press `Return`. The value in the `[ ]` is the current value that is not changed then.

Editing commands interactively

If you want to edit the whole request frame add a `edit_all=True` argument. In this case you can also change the command number, etc. on the fly.

```
In [1]: find_device()
Using first device with address 0x1337000023
Out[1]: 82527125539

In [2]: cmd1(edit_all=True)
Running Command 1
Enter value for HartFrame().delimiter.address_type (Enum of Flag) ['unique']>
Enter value for HartFrame().delimiter.number_of_expansion_bytes (BitField) [0]>
Enter value for HartFrame().delimiter.physical_layer_type (Enum of BitField) ['asynchronous']>
Enter value for HartFrame().delimiter.frame_type (Enum of BitField) ['stx']>
Enter value for HartFrame().address.master_address (Enum of Flag) ['primary_master']>
Enter value for HartFrame().address.burst_mode (Flag) [0]>
Enter value for HartFrame().command (UBInt8) [1]>
Enter value for HartFrame().byte_count (UBInt8) [0]>
Enter value for HartFrame().check_byte (CheckByte) [0]>
Request:
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'stx' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: 0 (Flag)
        address: 0 (BitField(38))
    expansion_bytes: Array():
    [
    ]
    command: 1 (UBInt8)
    byte_count: 0 (UBInt8)
    payload: Command1_ReadPrimaryVariableRequest():
    check_byte: 0 (CheckByte)

Response:
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: 0 (Flag)
        address: 82527125539 (BitField(38))
```

```
    expansion_bytes: Array():
    [
    ]
    command: 1 (UBInt8)
    byte_count: 7 (UBInt8)
    payload: Command1_ReadPrimaryVariableResponse():
        response_code: 'buffer_overflow' (UBInt8)
        field_device_status: non-primary_variable_out_of_limits (FieldDeviceStatus)
        units: 123 (UBInt8)
        primary_variable: 456.78900146484375 (BFloat)
    check_byte: 193 (CheckByte)

Out[2]: <htf.hart.protocol.HartFrame at 0x7fead57fabe0>
```

Passing contents to a command

The contents for interactively editable payload or frame contents can also be passed using the `inputs=[..]` argument.

To set the polling address to `2` non-interactively enter:

```
In [1]: find_device()
Using first device with address 0x1337000023
Out[1]: 82527125539

In [2]: cmd6(inputs=[2, 0])
Running Command 6
Request:
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'stx' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: 0 (Flag)
        address: 0 (BitField(38))
    expansion_bytes: Array():
    [
    ]
    command: 6 (UBInt8)
    byte_count: 0 (UBInt8)
    payload: Command6_WritePollingAddressRequest():
        polling_address: 2 (UBInt8)
        loop_current_mode: 0 (UBInt8)
    check_byte: 0 (CheckByte)
```

Enabling autocall

To enable the autocall-feature enter:

```
In [1]: %autocall 2
Automatic calling is: Full
```

Now you do not need to add brackets to the commands to be called:

```
In [1]: find_device
------> find_device()
Using first device with address 0x1337000023
Out[1]: 82527125539

In [2]: cmd1
------> cmd1()
Running Command 1
Request:
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'stx' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: 0 (Flag)
        address: 0 (BitField(38))
    expansion_bytes: Array():
    [
    ]
    command: 1 (UBInt8)
    byte_count: 0 (UBInt8)
    payload: Command1_ReadPrimaryVariableRequest():
    check_byte: 0 (CheckByte)

Response:
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: 0 (Flag)
        address: 82527125539 (BitField(38))
    expansion_bytes: Array():
    [
    ]
    command: 1 (UBInt8)
    byte_count: 7 (UBInt8)
    payload: Command1_ReadPrimaryVariableResponse():
        response_code: 'buffer_overflow' (UBInt8)
        field_device_status: non-primary_variable_out_of_limits (FieldDeviceStatus)
        units: 123 (UBInt8)
        primary_variable: 456.78900146484375 (BFloat)
    check_byte: 193 (CheckByte)
```

```
Out[2]: <htf.hart.protocol.HartFrame at 0x7f39eef2c3c8>
```

Setting the protocol factory

To use the protocol factory created for your device enter:

```
hart-shell -c /dev/ttyUSB0 -d protocol_factory.get_hart_frame_by_device_type
```

`hart-shell` needs a protocol factory that created a HART-frame by device type.

In the example `protocol_factory` is an importable python file and `get_hart_frame_by_device_type` is a method within it.

Scripting the HART-Shell

You can script the HART-Shell by putting your commands into a text file for easy reuse.

```
$ cat >hart_commands.txt <<EOL
find_device()
cmd1()
EOL
$ hart-shell -c /dev/ttyUSB0 < hart_commands.txt
```

You can even use the autocall-feature.

HART-Monitor

The `HartMonitor` enables you to debug HART communications. It can be run within a python script or via command line.

The `HartMonitor` support two types of decoders. One is used in the underlying `HartApplicationLayer` to decode frames in the first decoding phase. This is called the `decoder_generator`.

To be able to monitor multiple different devices with different device specific commands it is possible to do a second step decoding with the `protocol_factory`.

This is a factory method that returns a device specific HART-frame by device address.

```
class htf.hart.monitor.HartMonitor(comport: str | None = None, decoder_generator:
Type[HartFrame] | Callable[[...], HartFrame] | None = None, protocol_factory: Callable[[...],
Type[HartFrame]] | None = None, interactive: bool = False, argv: List[str] | None = None)
```

A HART Monitor that can be used interactively.

Parameters:

- **comport** – the comport to be used where the HART modem is connected to.
- **decoder_generator=None** – a callable that returns a HartFrame instance with or without device specific commands that is used to decode all frames (if `protocol_factory` is `None`).
- **protocol_factory=None** – a protocol factory that returns a device specific HART frame parser with respect to the supplied device address. For every received frame the frame is decoded using the `decoder_generator` that may be `None` and after that a device specific decoder is created respecting the device address.
- **interactive=True** – if set to `True` the HART Monitor becomes interactive
- **argv=None** – used for testing, etc.

```
handle_exception(frame: HartFrame | None, exc: BaseException) → None
```

The exception handler that prints the formatted output. This method can be overwritten.

This overwritten method may or may not call `raise` at the end to raise the exception again.

Parameters:

- **frame** – the HART frame to be monitored.
- **exc** – the exception that can be formatted.

```
handle_frame(frame: HartFrame) → None
```

The frame handler that prints the formatted output. This method can be overwritten.

Parameters:
frame – the HART frame to be monitored.

```
run(frame_count: int = 0) → None
```

Run the monitor and display received HART frames.

Parameters:
frame_count – The number of frames to be received. `0` means forever.

```
set_decoder_generator(decoder_generator: Type[HartFrame] | Callable[[...], HartFrame] |
None = None) → None
```

Set the decoder generator. If `decoder_generator` is `None` `HartFrame` is used.

Parameters:
decoder_generator=HartFrame – a parser for HART frames.

```
set_protocol_factory(protocol_factory: Callable[[...], Type[HartFrame]] | None) → None
```

Set the protocol factory.

Parameters:
protocol_factory=None – a protocol factory that returns a device specific HART frame parser with respect to the supplied device address.

Command line usage

`htf` comes with a HART-Monitor command line utility called `hart-monitor`.

To start it enter

```
$ hart-monitor -c <comport>
```

Show help

```
$ hart-monitor -h
usage: hart-monitor [-h] -c COMPORT [-d Protocol database] [-V]

hart-monitor - HILSTER Testing Framework (HILSTER GmbH, http://hilster.io)

optional arguments:
  -h, --help            show this help message and exit
  -c COMPORT, --comport COMPORT
                        comport for HART communication
  -d Protocol database, --database Protocol database
                        An import string to the protocol database where device
                        specific HART frames can be found. See docs for
                        details.
  -V, --version         Show version and exit
```

Setting the protocol factory

To use the protocol factory created for your device enter:

```
hart-monitor -c /dev/ttyUSB0 -d protocol_factory.get_hart_frame_by_device_type
```

`hart-monitor` needs a protocol factory that created a HART-frame by device type.

In the example `protocol_factory` is an importable python file and `get_hart_frame_by_device_type` is a method within it.

Customizing the HART-Monitor

If a different output format is needed etc. the class `HartMonitor` can be subclassed and `handle_frame` can be overwritten.

In the following example every received frame is serializied to be printed on a single line with a timestamp.

```
from htf.hart.monitor import HartMonitor
import datetime


class MyHartMonitor(HartMonitor):
    def handle_frame(self, frame):
        frame_type = frame.delimiter.frame_type.get()
        direction = " ->"
        if frame_type == "ack":
            direction = "<- "
        command = frame.command.get()
        byte_count = frame.byte_count.get()
        address = frame.address.address.get()
        payload = str(frame.payload).replace("\n", "").replace("    ", ", ")
        print(str(datetime.datetime.utcnow()) +
              ": " + "%s %3i %s 0x%010x %3i (%s)" %
              (frame_type.upper(),
               command,
               direction,
               address,
               byte_count,
               payload))

if __name__ == "__main__":
    hm = MyHartMonitor(comport="COM46")
    hm.run(n=0)
```

HART Device Communication

The `HartDeviceCommunication` simplifies the communication with a HART device.

It can be either used as an instance or as a context block and helps to find a device and to query commands on that device.

```
class htf.hart.device_communication.HartDeviceCommunication(comport: str,
polling_addresses: List[int] = [0], device_address: int | None = None, decoder_generator:
Type[HartFrame] | Callable[[...], HartFrame] | None = None)
```

A communication to a HART device usable as a context block.

Parameters:

- **comport** – the comport to be used where the HART modem is connected to.
- **polling_addresses=[0]** – the polling addresses used to find a HART device.
- **device_address=None** – the initial HART device address. If set to an interger no device must be looked up.
- **decoder_generator=None** – a callable that returns a HartFrame instance with or without device specific commands.

```
close() → None
```

Close the HART application layer and close the HART interface supplied. Is called in __del__, too.

```
find_device(polling_addresses: List[int] | None = None) → int
```

Find the first HART device by polling all addresses stated in `polling_addresses` and use the first device found.

If no device is found an `AssertionError` is raised.

Parameters:
polling_addresses=[0] – the polling addresses used to find a HART device.
Returns:
Unique HART address of the first device found in polling.
Return type:
int

```
query(request: HartFrame, number_of_preambles: int = 5, preamble: bytes = b'\xff',
timeout: int | float = 1.0) → HartFrame | None
```

Query a HART command. The HART frame is encoded automatically.

Parameters:

- **request** – the hart frame to be sent.
- **number_of_preambles=5** – the number of preambles.
- **preamble='ÿ'** – `number_of_preambles` times `preamble` is sent before the frame content is sent.
- **timeout=1.0** – the timeout in seconds.

Returns:
if no frame was received `None` is returned.
Return type:
HartFrame or None

```
set_device_address(device_address: int) → None
```

Set the current HART device address.

Parameters:
device_address – the HART device address to be used (<= 38 bits)

In the following examples `Command 1` is queried on the first found device.

Usage with an instance:

```
from htf.hart import HartDeviceCommunication
from htf.hart.protocol import HartFrame
```

```
dc = HartDeviceCommunication(comport="/dev/ttyUSB1")
dc.find_device()

82527125539
cmd1_request = HartFrame(command=1)
cmd1_response = dc.query(cmd1_request)
print(cmd1_response)
    HartFrame():
        delimiter: Delimiter():
            address_type: 'unique' (Flag)
            number_of_expansion_bytes: 0 (BitField(2))
            physical_layer_type: 'asynchronous' (BitField(2))
            frame_type: 'ack' (BitField(3))
        address: Address():
            master_address: 'primary_master' (Flag)
            burst_mode: 0 (Flag)
            address: 82527125539 (BitField(38))
        expansion_bytes: Array():
        [
        ]
        command: 1 (UBInt8)
        byte_count: 7 (UBInt8)
        payload: Command1_ReadPrimaryVariableResponse():
            response_code: 'buffer_overflow' (UBInt8)
            field_device_status: non-primary_variable_out_of_limits (FieldDeviceStatus)
            units: 123 (UBInt8)
            primary_variable: 456.78900146484375 (BFloat)
        check_byte: 193 (CheckByte)

dc.close()
```

If a `device_address` is supplied `find_device` needs not to be called.

When used as a context block and no `device_address` is supplied `find_device` is called automatically. When the context block is left the underlying HART interface and Application Layer are closed automatically.

Example usage with a context block:

```
from htf.hart import HartDeviceCommunication
from htf.hart.protocol import HartFrame


with HartDeviceCommunication(comport="/dev/ttyUSB1") as dc:
    dc.find_device()
    cmd1_request = HartFrame(command=1)
    cmd1_response = dc.query(cmd1_request)
    print(cmd1_response)

82527125539
HartFrame():
    delimiter: Delimiter():
        address_type: 'unique' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'ack' (BitField(3))
    address: Address():
```

```
        master_address: 'primary_master' (Flag)
        burst_mode: 0 (Flag)
        address: 82527125539 (BitField(38))
    expansion_bytes: Array():
    [
    ]
    command: 1 (UBInt8)
    byte_count: 7 (UBInt8)
    payload: Command1_ReadPrimaryVariableResponse():
        response_code: 'buffer_overflow' (UBInt8)
        field_device_status: non-primary_variable_out_of_limits (FieldDeviceStatus)
        units: 123 (UBInt8)
        primary_variable: 456.78900146484375 (BFloat)
    check_byte: 193 (CheckByte)
```

HART Fuzzing

The HART Testing Framework lets you easily run fuzzing tests on your HART device since it is based on OSER. You should have read the OSER documentation.

In the first example all HART frames shall have a valid checkbyte at the end. All commands from `0` to `255` shall be sent to the target. The payload is the default.

```
from htf.hart import HartFrame
import oser

h = HartFrame(0)

h.command.set_fuzzing_values(range(256))

for frame in h.fuzzing_iterator():
    encoded = frame.encode()  # update checkbyte
    print(frame)
    print(oser.to_hex(encoded))

HartFrame():
    delimiter: Delimiter():
        address_type: 'polling' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'stx' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: 0 (Flag)
        address: 0 (BitField(6))
    expansion_bytes: Array():
    [
    ]
    command: 0 (UBInt8)
    byte_count: 0 (UBInt8)
    payload: Command0_ReadUniqueIdentifierRequest():
    check_byte: 130 (CheckByte)

   0|  1|  2|  3|  4
\x02\x80\x00\x00\x82
HartFrame():
```

```
    delimiter: Delimiter():
        address_type: 'polling' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'stx' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: 0 (Flag)
        address: 0 (BitField(6))
    expansion_bytes: Array():
    [
    ]
    command: 1 (UBInt8)
    byte_count: 0 (UBInt8)
    payload: Command1_ReadPrimaryVariableRequest():
    check_byte: 131 (CheckByte)

# [%< %< %< %< %< %< %< %< %< %<]

   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23
\x02\x80\xfd\x13\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x6c
HartFrame():
    delimiter: Delimiter():
        address_type: 'polling' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'stx' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: 0 (Flag)
        address: 0 (BitField(6))
    expansion_bytes: Array():
    [
    ]
    command: 254 (UBInt8)
    byte_count: 19 (UBInt8)
    payload: b''
    check_byte: 111 (CheckByte)

   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23
\x02\x80\xfe\x13\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x6f
HartFrame():
    delimiter: Delimiter():
        address_type: 'polling' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'stx' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: 0 (Flag)
        address: 0 (BitField(6))
    expansion_bytes: Array():
    [
    ]
    command: 255 (UBInt8)
    byte_count: 19 (UBInt8)
    payload: b''
    check_byte: 110 (CheckByte)
```

```
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23
\x02\x80\xff\x13\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x6e
```

In the second example the whole payload is random with a random length. All frames have a valid length field and a valid checkbyte.

```
from htf.hart import HartFrame

import oser
import string

h = HartFrame(0)

_random_length_iterator = oser.RandomIntegerFuzzingValue(a=0, b=40, count=1)

h.payload = oser.String(length=lambda self: next(_random_length_iterator()))
h.payload.set_fuzzing_values(
     oser.RandomBytesFuzzingValue(length=40, count=5)
)

for frame in h.fuzzing_iterator():
    encoded = frame.encode()  # update checkbyte
    print(frame)
    print(oser.to_hex(encoded))

HartFrame():
    delimiter: Delimiter():
        address_type: 'polling' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'stx' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: 0 (Flag)
        address: 0 (BitField(6))
    expansion_bytes: Array():
    [
    ]
    command: 0 (UBInt8)
    byte_count: 28 (UBInt8)
    payload: b'\x8c\xdd\x0b]\x0cd\xf9\xde\x1a1fo\xea{@n\x82\x94dG
\xf3\x9e\xbb`\x06\x83`\xe5L\xa7\xdb?q*?\xbeW\x08\x8c'
    check_byte: 150 (CheckByte)

   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25| 26| 27| 28| 29| 30| 31| 32| 33| 34| 35| 36| 37| 38
\x02\x80\x00\x1c\x8c\xdd\x0b\x5d\x0c\x64\xf9\xde\x1a\x31\x66\x6f\xea\x7b\x40\x6e\x82\x94\x64\x47\x20\xf
HartFrame():
    delimiter: Delimiter():
        address_type: 'polling' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'stx' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: 0 (Flag)
        address: 0 (BitField(6))
```

```
    expansion_bytes: Array():
    [
    ]
    command: 0 (UBInt8)
    byte_count: 34 (UBInt8)
    payload: b'\xc4OHo\x17\xcb\xaf\xf5\x9b\xd0[N\x88n\x9b\x013\xa4z\xf2\\
\x15\xa8\xcb\xd9;2\x15\x10H\x7fC\xf3\xbd~\xf2&>\xab$'
    check_byte: 235 (CheckByte)

   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25
\x02\x80\x00\x22\xc4\x4f\x48\x6f\x17\xcb\xaf\xf5\x9b\xd0\x5b\x4e\x88\x6e\x9b\x01\x33\xa4\x7a\xf2\x5c\xe
HartFrame():
    delimiter: Delimiter():
        address_type: 'polling' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'stx' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: 0 (Flag)
        address: 0 (BitField(6))
    expansion_bytes: Array():
    [
    ]
    command: 0 (UBInt8)
    byte_count: 24 (UBInt8)
    payload: b'w]w_\xeb\x1b\x9f\xad\t\xcd\x9ei\xf7te\x17\x0c\x96\x9e\xe0/
\x9f\x1cR\xd7\xeb\x87\x9f\xfc\x1aC^q\xed/\x085\xfc\xaf6'
    check_byte: 182 (CheckByte)

   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25| 26| 27| 28| 29| 30| 31| 32| 33| 34| 35| 36| 37| 38| 39| 40| 41| 42| 43| 44
\x02\x80\x00\x18\x77\x5d\x77\x5f\xeb\x1b\x9f\xad\x09\xcd\x9e\x69\xf7\x74\x65\x17\x0c\x96\x9e\xe0\x2f\x9
HartFrame():
    delimiter: Delimiter():
        address_type: 'polling' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'stx' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: 0 (Flag)
        address: 0 (BitField(6))
    expansion_bytes: Array():
    [
    ]
    command: 0 (UBInt8)
    byte_count: 25 (UBInt8)
    payload: b'\x9e\x8dR\xebO\xb8\xa3\xe8\x0b|c\x02,>GpFD4\xdc\xed\xe9,
\xa6\xbd\x1f\xa2;`\xd3\xb5\x83Nv\xe7\x8a\xcc\xd7\xbeW'
    check_byte: 225 (CheckByte)

   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25| 26| 27| 28| 29| 30| 31| 32
\x02\x80\x00\x19\x9e\x8d\x52\xeb\x4f\xb8\xa3\xe8\x0b\x7c\x63\x02\x2c\x3e\x47\x70\x46\x44\x34\xdc\xed\xe
HartFrame():
    delimiter: Delimiter():
        address_type: 'polling' (Flag)
        number_of_expansion_bytes: 0 (BitField(2))
```

```
        physical_layer_type: 'asynchronous' (BitField(2))
        frame_type: 'stx' (BitField(3))
    address: Address():
        master_address: 'primary_master' (Flag)
        burst_mode: 0 (Flag)
        address: 0 (BitField(6))
    expansion_bytes: Array():
    [
    ]
    command: 0 (UBInt8)
    byte_count: 32 (UBInt8)
    payload: b"\x1c`]
\xbain'\x8dq\xcb\x0fw\xa3\x9b\x89\xb8T\xe4&\x08\x90{\xfa\x9f\x08\xb2\xbe\xd5\xec\x12\xe7\xf2b&\x943Yc\x
    check_byte: 165 (CheckByte)

   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13| 14| 15| 16| 17| 18| 19| 20| 21| 22| 23|
24| 25| 26| 27| 28| 29| 30| 31| 32| 33| 34| 35| 36| 37| 38| 39
\x02\x80\x00\x20\x1c\x60\x5d\xba\x69\x6e\x27\x8d\x71\xcb\x0f\x77\xa3\x9b\x89\xb8\x54\xe4\x26\x08\x90\x7
```

HART Slave Simulator

The `HartSlaveSimulator` allows simulation of HART slaves by sub-classing the `HartSlaveSimulator`.

```
class htf.hart.slave_simulator.HartSlaveSimulator(comport: str, decoder_generator:
Type[HartFrame] | Callable[[...], HartFrame] | None = None, polling_address: int = 0,
device_type: int = 0, device_id: int = 0)
```

```
close() → None
```

Closes the HART Application Layer of the simulator.

```
handle_command0(frame: HartFrame) → HartFrame | None
```

Handler for command 0 HART requests. Checks if the polling address in the frame matches the one set in the simulator.

Parameters:
frame – the command 0 HART request
Returns:
the response containing device type, device id and protocol revision
Return type:
HartFrame

```
handle_default(frame: HartFrame) → None
```

Default handler for HART requests that don't have a specialized handler.

Parameters:

frame – the HART frame to handle

```
handle_exception(exc: BaseException) → None
```

The exception handler that prints the raised exception.

Parameters:
exc – the exception that was raised.

```
handle_frame(frame: HartFrame) → None
```

Checks that the device address matches and whether responses to the HART command are disabled. Calls possible frame callbacks and sends a HART response frame if there is one.

Parameters:
frame – the HART frame to handle

```
register_frame_callback(callback: Callable[[float, HartFrame], None]) → None
```

Registers a callback for every HART request and response that is received or send out. Callbacks are called with two arguments: a timestamp and the HART frame.

Parameters:
callback – callback for HART frames

```
run(frame_count: int = 0) → None
```

Run the HART device simulator

Parameters:
frame_count – The number of frames to handle. `0` means forever.

```
run_while(condition: bool | Callable[[...], bool]) → None
```

Simulate while the condition is `True`.

Parameters:
condition – the condition for the run.

```
set_response_enabled(command: int | str, enabled: bool = True) → None
```

Enables/disables responses to a given HART command number. Responses to all commands are enabled by default.

Parameters:

- **command** – HART command number or “any”
- **enabled=True** – enabled state

```
unregister_frame_callback(callback: Callable[[float, HartFrame], None]) → None
```

Unregisters a previously registered callback.

Parameters:
callback – callback for HART frames

If you’d like to add other handlers for different commands simply add a handler method that fits the name convention `handle_command<cmd>` where `<cmd>` is the desired command number.

```
from htf.hart.slave_simulator import HartSlaveSimulator
from htf.hart import HartFrame, ResponseCode, FieldDeviceStatus
import oser

# Command abstractions
class Command123_ReadValueRequest(oser.ByteStruct):
    def __init__(self):
        super(Command123_ReadValue, self).__init__()
        self.double_value = oser.Flag(False)

class Command123_ReadValueResponse(oser.ByteStruct):
    def __init__(self):
        super(Command123_ReadValue, self).__init__()
        self.response_code = ResponseCode()
        self.field_device_status = FieldDeviceStatus()
        self.value = oser.UBInt8()

# Decoder generator extending the default HART frame implementation
class MyHartDeviceFrame(HartFrame):

    def requests_generator(self):
        for command, parser in super(MyHartDeviceFrame, self).requests_generator():
            yield command, parser
        yield 123, oser.Lazy(Command123_ReadValueRequest)

    def responses_generator(self):
        for command, parser in super(MyHartDeviceFrame, self).responses_generator():
            yield command, parser
        yield 123, oser.Lazy(Command123_ReadValueResponse)

# Slave simulator with the extended HART frame (MyHartDeviceFrame)
class MyHartDevice(HartSlaveSimulator):

    # Initialization of the device attributes
    def __init__(self, comport):
            super(MyHartDevice, self).__init__(comport, decoder_generator=MyHartDeviceFrame,
                                               polling_address=0, device_type=0x1337,
                                               device_id=0x23)
        self.value = 321

    # Implementation of the response behavior
```

```
    def handle_command123(self, request):
        response = self._create_response(request)

        if request.payload.double_value.get():
            response.payload.value.set(self.value * 2)
        else:
            response.payload.value.set(self.value)

        return response

if __name__ == "__main__":
    sim = MyHartDevice("COM1")
    sim.run(10)
```

Changelog

htf-hart-3.2.0

- ship `cp312-abi3` wheels that are compatible with Python 3.12 and newer versions

htf-hart-3.1.6

- implement Cyber Resilience Act Conformity (CRA)
- update dependencies

htf-hart-3.1.5

- add support for Python 3.13

htf-hart-3.1.4

- add requirements for validation purposes

htf-hart-3.1.3

- add build environment data for validation purposes

htf-hart-3.1.2

- `htf-hart` can now be used standalone

htf-hart-3.1.1

- add support for Python 3.12

htf-hart-3.1.0

- add type-hints
- use `hlm-3.1`
- use `oser-3.1`

htf-hart-3.0.1

- add support for Python 3.11

htf-hart-3.0.0

- extract `htf-hart` from `htf`

Communication

This chapter describes the modules for the asynchronous communication framework.

Pluggable Notifier

The `htf.communication.PluggableNotifier` is an observer that lets subjects register tuples of `name`, `filterExpression` and a `callback`.

When data is input all registered subjects are used the following way. First the `filterExpression` is called with the data as a parameter. If `True` is returned the callback is called with the data as a parameter.

With the `htf.communication.PluggableNotifier` you can create an asynchronous communication framework easily.

```
class htf.communication.PluggableNotifier(verbosity: int = 0,
raise_exceptions: bool = False, filter_exception_handler: Callback | None = None,
callback_exception_handler: Callback | None = None)
```

`PluggableNotifier` implements an observer pattern. Subjects can register tuples consisting of executable filter expressions and callbacks to be notified about events.

There is a method to enqueue new items and a thread to apply filters on items. Communication between method and thread is done via a queue.

`PluggableNotifier` extends `threading.Thread`.

Parameters:

- **verbosity** – if set to 1 debug messages are printed to stdout
- **raise_exceptions** – if set to `True` exceptions are raised if a filter expression or a callback raises an exception. This may be used for debugging. In case of an exception `stop` is called so no further items can be put in.
- **filter_exception_handler** – a callback that is called to handle a raised exception in a filter expression. If the `filter_exception_handler` returns `True` the exception is not raised afterwards. Else the exception handler raises the catched exception and stopps the thread. The supplied handler musst accept `name` and `item` arguments. `name` is the callback name supplied by `register` and `item` is the supplied item from `input`.

- **callback_exception_handler** – a callback that is called to handle a raised exception in a callback. If the `callback_exception_handler` returns `True` the exception is not raised afterwards. Else the exception handler raises the catched exception and stopps the thread. The supplied handler musst accept `name` and `item` arguments. `name` is the callback name supplied by `register` and `item` is the supplied item from `input`.

```
get_filter_names() → List[str]
```

Get the filter names.

Returns:
a sorted list containing the filter names.
Return type:
list of str

```
input(item: Any) → None
```

Input an `item` to be used with all registered subjects.

Parameters:
item – an item to be processed.
Raises:

- **RuntimeError** – if `stop`
- **was called before** –

```
join(timeout: float | None = 1.0) → None
```

Wait until the thread ends.

Parameters:
timeout – the timeout in seconds. Default: 1.0 seconds.

```
register(name: str, filter_expression: Callback, callback: Callback) → None
```

Register a tuple of *name*, *filter_expression* and a *callback*.

Parameters:

- **name** – a name for the tuple (for debugging, see `verbosity`).
- **filter_expression** – a callable filter expression (see `htf.filters` for more information) or a method that is called. `filter_expression` is called with the `item` from `input` as a parameter. If `filter_expression` returns *True* the callback is called with the same *item*.
- **callback** – a callable object that is called in case `filter_expression(item)` returns `True`.

```
run() → None
```

Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

```
stop() → None
```

Stop the thread.

```
unregister(name: str) → None
```

Unregister a callback with a given `name`.

Parameters:
name – the name of the callback to be unregistered.
Raises:
KeyError – if `name` was not found in `notifications`.

Notification Callbacks

Notification callbacks can be used in `htf.communication.PluggableNotifier` as the *callback*.

For example you can enqueue an item into a queue the callee just created to asynchronously receive data.

Example:

```
from htf.communication import PluggableNotifier, enqueueItem
import queue # or Queue in Python 2

pn = PluggableNotifier()
q = queue.Queue()
pn.register("subject", filterExpression=lambda *args,
            **kwargs: kwargs["item"]=="example item", callback=enqueueItem(q))
pn.input("example item") # is passed
item = q.get(block=True, timeout=1)
print(item) # "example item"
pn.input("another item") # is filtered out
item = q.get(block=True, timeout=1) # raises Queue.Empty
```

```
htf.communication.notification_callbacks.enqueue_item(queue: Queue) → Callable[[str, Any],
None]
```

Enqueues `item` into `queue`. This way asynchronous communication can be set up easily.

Parameters:
queue (*queue.Queue*) – a queue to be used.

Returns:
an anonymous method that can be called with an `item` that is put into `queue`.
Return type:
callable

SSH Client

With the SSH Client you can run commands remotely via SSH.

```
class htf.communication.SSHClient
```

`SSHClient` executes commands remotely using ssh.

```
close() → None
```

Close current ssh connection.

```
connect(hostname: str, username: str | None = None, password: str | None = None) → None
```

Establish an ssh connection.

Parameters:

- **hostname** – the hostname or ip address
- **username=None** – The username to be used. If set to `None` the current username is used instead of using the ssh keys in your system.
- **password=None** – The password for `username`. If set to `None` the password for `username` or the current username is used.

```
execute(*command: str) → Tuple[int, bytes, bytes]
```

Execute a command remotely via ssh.

Parameters:
***command** – the command to be executed.
Returns:
a tuple containing exitcode, stdout and stderr of the executed command.
Return type:
tuple

SFTP Client

With the SFTP Client it is possible to do file operations remotely via SSH.

```
class htf.communication.SFTPClient
```

`SFTPClient` lets you access files via sftp protocol.

```
close() → None
```

Close current sftp connection.

```
connect(hostname: str, username: str | None = None, password: str | None = None) → None
```

Establish an sftp connection.

Parameters:

- **hostname** – the hostname or ip address
- **username=None** – The username to be used. If set to `None` the current username is used instead of using the ssh keys in your system.
- **password=None** – The password for `username`. If set to `None` the password for `username` or the current username is used.

```
get(remotepath: str, localpath: str) → None
```

Copies file identified by `remotepath` to `localpath` using sftp protocol.

Parameters:

- **remotepath** – the remote file to be copied
- **localpath** – the local path where the field is copied to

```
put(localpath: str, remotepath: str) → SFTPAttributes
```

Copies file identified by `localpath` to `remotepath` using sftp protocol.

Parameters:

- **localpath** – the local path to be copied
- **remotepath** – the remote file where the field is copied to

Returns:
the result of os.stat
Return type:
SFTPAttributes

```
remove(remotepath: str) → None
```

Removes a file identified by `remotepath`.

Parameters:
remotepath – the path for the file to be removed
Raises:
IOError – if the path is not a file

```
rename(oldpath: str, newpath: str) → None
```

Remotely renames a file or a folder.

Parameters:

- **oldpath** – the file or folder to be renamed
- **newpath** – the new file- or foldername

Raises:
IOError – if something goes wrong

```
unlink(remotepath: str) → None
```

Removes a file identified by `remotepath`.

Parameters:
remotepath – the path for the file to be removed
Raises:
IOError – if the path is not a file

SLIP — Serial Line Internet Protocol

SLIP contains a mixin to be used to implement SLIP-based communications.

If you don't know about SLIP you should first read RFC 1055.

The `htf.communication.SlipMixin` contains two methods for encoding and decoding SLIP-encoded communication frames.

```
class htf.communication.SlipMixin
```

The `SlipMixin` helps to encode and decode frames transferred in the Serial Line Internet Protocol.

It cannot be used standalone and must be mixed into a class that implements the `htf.communication.SlipInterface`.

```
slip_receive_frame(timeouts: int | None = None, max_length: int = 1024) → bytes | None
```

Receive a frame encoded in the Serial Line Internet Protocol and return the decoded frame.

Parameters:

- **timeouts=None** – the number of timeouts until None is returned. The timeout depends on the mixing class and underlying communication system.
- **max_length=1024** – maximum number of bytes that will be received, not including ESC. Prevents the method from perpetually blocking.

Returns:
the decoded frame if a frame was received or
`None` if no frame was received.

Return type:
bytes or None

```
slip_send_frame(frame: bytes) → None
```

Send a frame encoded in the Serial Line Internet Protocol.

Parameters:
frame – the frame to be sent.

A class can subclass `htf.communication.SlipMixin` and must implement the `htf.communication.SlipInterface`.

```
class htf.communication.SlipInterface
```

```
receive_byte() → bytes
```

Receive a single byte. If no byte was received, `None` is returned to tell about a timeout.

Returns:
a single byte that was received or `None` **if a**
timeout occurred.

Return type:
bytes or None

```
send_byte(byte: bytes) → None
```

Send a single byte.

Parameters:
byte – a single byte to be sent.

In the following example one-way SLIP communication is established with a client and a server.

SLIP-Client:

```
from htf.communication import SlipMixin
import serial


class SlipClient(SlipMixin):
    def __init__(self, comport):
        self._serial = serial.Serial(comport, 115200,
                                     timeout=1.0, rtscts=False)

    def send_byte(self, byte):
        self._serial.write(byte)

    def receive_byte(self):
        return self._serial.read(1)

    def send(self, message):
        if not isinstance(message, bytes):
            message = message.encode()
        self.slip_send_frame(message)

if __name__ == "__main__":
    s = SlipClient("/dev/ttyUSB1")
    message = "Hello World!"
    s.send(message)
```

SLIP-Server:

```
from htf.communication import SlipMixin
import serial


class SlipServer(SlipMixin):
    def __init__(self, comport):
        self._serial = serial.Serial(comport, 115200,
                                     timeout=1.0, rtscts=False)

    def send_byte(self, byte):
        self._serial.write(byte)

    def receive_byte(self):
        return self._serial.read(1)

    def send(self, message):
        if not isinstance(message, bytes):
            message = message.encode()
        self.slip_send_frame(message)

    def receive(self):
        message = self.slip_receive_frame(timeouts=3)
        print("Received message:", message)

if __name__ == "__main__":
    s = SlipServer("/dev/ttyUSB2")
    while True:
        s.receive()
```

Changelog

htf-communication-4.1.0

- ship `cp312-abi3` wheels that are compatible with Python 3.12 and newer versions

htf-communication-4.0.7

- update dependencies

htf-communication-4.0.6

- implement Cyber Resilience Act Conformity (CRA)
- update dependencies

htf-communication-4.0.5

- add support for Python 3.13

htf-communication-4.0.4

- add requirements for validation purposes

htf-communication-4.0.3

- add build environment data for validation purposes

htf-communication-4.0.2

- `htf-communication` can now be used standalone

htf-communication-4.0.1

- add support for Python 3.12

htf-communication-4.0.0

- remove all camel-case methods
- add type-hints
- use `hlm-3.1`
- use `oser-3.1`

htf-communication-3.0.1

- add support for Python 3.11

htf-communication-3.0.0

- extract `htf-communication` from `htf`

I/O — Input and Output

Xentara — Real-Time Automation Platform

Introduction

Xentara is a modular, secure and powerful Real-Time Automation Platform. You can find more information at the Xentara website.

`htf` includes a websocket-client to interact with the Xentara runtime. Thus, users can for example deploy tests on their machines and execute tests remotely.

The Xentara-client is async-only.

Open a Connection

To open a connection, instantiate `XentaraClient` and await it.

```
client = await XentaraClient(
    url="wss://xentara-host/api/ws",
    root_element="Hardware",
)
```

> **Note**
>
> Don't forget the `await` statement!

Alternatively use `create`.

```
client = await XentaraClient.create(
    url="wss://xentara-host/api/ws",
    root_element="Hardware",
)
```

The root element (`root_element`) must match the configuration Xentara was started with. The contents are browsed and fetched automatically on instantiation.

Authentication

The Xentara-client supports basic authentication.

The credentials are simply added to the url, i.e.:

```
client = await XentaraClient(
    url="wss://<user>:<password>@xentara-host/api/ws",
)
```

where `<username>` is the username and `<password>` is the password to be used.

TLS/SSL

Use `wss://` in the url for a TLS/SSL secured connection and `ws://` for an unsafe and unencrypted connection.

Self-signed certificates

To disable the TLS/SSL certificate verification, just add `verify_ssl=False`.

```
client = await XentaraClient(
    url="wss://<user>:<password>@xentara-host/api/ws",
    root_element="Hardware",
    verify_ssl=False,
)
```

Close a Connection

To close an open connection, simply run:

```
await client.close()
```

Reading values

To read a value, call `read`:

```
value = await client.read("<element>")
```

`<element>` is a descriptor of a Xentara element, for example `"Hardware.EtherCAT Bus.EL2624.Output 1"`.

In addition, the Xentara-client can be used like a dictionary to read values.

```
value = await client["<element>"].read()
```

Writing values

To write a value, call `write`:

```
await client.write("<element>", value)
```

`<element>` is a descriptor of a Xentara element, for example `"Hardware.EtherCAT Bus.EL2624.Output 1"`.

The Xentara-client can be used like a dictionary to write values.

```
await client["<element>"].write(value)
```

Subscriptions

To subscribe to changes, use the `subscribe` function that returns a `Subscription`.

The `htf.xentara.Subscription` returns an async iterable that can be used in different ways. Each iteration yields an `Event`.

The easiest way to use a subscription is:

```
async with client.subscribe("<element>") as subscription:
    for i in range(5):
        event = await subscription.receive()
        value = event.value
        print(f"{i}: {event=}")
```

`aiter` can also be used explicitly:

```
async with client.subscribe("<element>") as subscription:
    iterator = aiter(subscription)
    for i in range(5):
        event = await iterator.__anext__()
        value = event.value
        print(f"{i}: {event=}")
```

For an infinite number of received values, `async for` can be used.

```
subscription = client.subscribe("<element>")

async with subscription as subscription:
    async for event in subscription:
        value = event.value
        print(f"{i}: {event=}")
```

String Representation

The Xentara client can be printed to get information about it.

```
print(client)
```

Use the Client like a Dictionary

To get the available elements, the `XentaraClient` can be used like a dictionary.

```
keys = client.keys()
values = client.values()
items = client.items()
length = len(client)
```

XentaraClient

```
class htf.xentara.XentaraClient(url: str, root_element: str = 'Hardware', verify_ssl: bool =
True)
```

The Xentara websocket client.

Parameters:

- **url** – The url to connect to the Xentara server.
- **root_element** – The root element to browse for elements. This is set in the Xentara configuration.
- **verify_ssl** – if set to `False` TLS/SSL certificate verification is switched off. Default is `True`.

```
async close() → None
```

Close the connection.

```
async copy() → XentaraClient
```

Return a copy of the instance with a new websocket connection.

```
async classmethod create(url: str, root_element: str = 'Hardware', verify_ssl: bool =
True) → XentaraClient
```

Create the Xentara client. Is called when the class instance is awaited, too.

Parameters:

- **url** – The url to connect to the Xentara server.
- **root_element** – The root element to browse for elements. This is set in the Xentara configuration.
- **verify_ssl** – if set to `False` TLS/SSL certificate verification is switched off. Default is `True`.

```
async read(element: str) → Any
```

Read the current value.

Returns:
the current value
Return type:
Any

```
subscribe(element: str, maximum_update_interval: int = 500, minimum_update_interval: int
= 1000) → Subscription
```

Create a subscription.

Parameters:

- **element** – The element to be subscribed.
- **maximum_update_interval** – the maximum update interval in milliseconds (default: 500)
- **minimum_update_interval** – the minimum update interval in milliseconds (default: 1000)

Returns:
the instance of the subscription
Return type:
Subscription

```
async write(element: str, value: Any) → Any
```

Write a value.

Parameters:
value – the value to be written

Element

```
class htf.xentara.Element(client: XentaraClient, name: str)
```

An element to read and write values. Used when :class:~`htf.xentara.XentaraClient` is used like a dictionary.

```
async read() → Any
```

Read the current value.

Returns:
the current value
Return type:
Any

```
async write(value: Any) → Any
```

Write a value.

Parameters:
value – the value to be written

Subscription

```
class htf.xentara.Subscription(client: XentaraClient, element: str, maximum_update_interval:
int = 500, minimum_update_interval: int = 1000)
```

Subscriptions for Xentara elements.

Users will never call this method directly.

Parameters:

- **client** – the current Xentara client's instance (passed automatically)
- **element** – the name of the element the subscription is for
- **maximum_update_interval** – the maximum update interval in milliseconds (default: 500)
- **minimum_update_interval** – the minimum update interval in milliseconds (default: 1000)

```
async receive() → Event
```

Receive the next event in the subscription.

```
async start() → None
```

Start the subscription.

```
async stop() → None
```

Stop the subscription.

Event

```
class htf.xentara.Event(timestamp: float, name: str, uuid: UUID, value: Any)
```

Yielded in `Subscription` when used as an async iterator.

```
name: str
```

the name of the element

```
timestamp: float
```

the timestamp as a float

```
uuid: UUID
```

the uuid of the element

```
value: Any
```

the current value of the element

LabJack — DAQ devices

Introduction

htf offers native support for LabJack's DAQ devices to easily use them in tests without the need of a driver (except on Windows, where a driver is necessary).

Limitations

Streaming is not yet implemented.

Preparation on Linux

Simply install `libusb` and everything works out of the box, even from within Docker.

Driver installation on Windows

The easiest way to use LabJack devices via USB is to install the LJM software package that includes an USB driver. It can be downloaded here.

Alternatively you can use libusb as a driver for USB communication. To do that, please follow the installation instructions from `libusb`.

Supported devices

htf supports the following LabJack devices

- T4
- T7
- T7-Pro
- T8

List USB devices

To list connected USB devices, run

```
labjack list
# 0cd5:0007 (bus 1, device 76): LabJack T7 'My_T7_123' Ethernet: 192.168.0.100 Wifi: <not connected>
```

The output lists all connected devices with the device name and IP addresses. It does not search for devices via network.

Opening USB devices

To open any USB device, do not pass any parameters when instantiating a client.

```
from htf.labjack import T7
labjack_device = T7()
print(labjack_device)
# <T7 connection='Modbus TCP' name='My_T7_1896' host=192.168.0.100 port=502>
```

To open a USB device explitly pass the device name.

```
from htf.labjack import T7
labjack_device = T7(name="My_T7_1234")  # use your device's name instead
print(labjack_device)
# <T7 connection='USB' name='My_T7_1234'>
```

Opening Modbus TCP devices

To open a device via Modbus TCP use the ip address or host name when instantiating the client.

```
from htf.labjack import T7
```

```
labjack_device = T7(host="my-labjack-device")
print(labjack_device)
# <T7 connection='Modbus TCP' name='My_T7_1234' host='my-labjack-device' port=502>
# or
labjack_device = T7(host="192.168.0.100")
print(labjack_device)
# <T7 connection='Modbus TCP' name='My_T7_1234' host='192.168.0.100' port=502>
```

Interaction with the devices

All devices have a respective class named after the device name, which provides attributes that can be read or written to to access Modbus registers. In addition, convenience functions are provided to improve ease of use.

The attributes have the same name as the Modbus registers, but in lower case.

For example the following code reads the current value from `AIN0` as `float`. The corresponding Modbus register is read automatically.

```
from htf.labjack import T7

labjack_device = T7(host="my-labjack-device")
current_ain: float = labjack_device.ain0
```

To output `3 V` on `DAC0`, simply run:

```
from htf.labjack import T7

labjack_device = T7(host="my-labjack-device")
labjack_device.dac0 = 3.0
```

Transactions

By default all communication with a LabJack devices is locked (thread-safe). Additionally, all convenience functions use another lock to assure that multiple commands are run without an interruption.

To run multiple commands from user code, use a with statement:

```
from htf.labjack import T7

labjack_device = T7(host="my-labjack-device")
with labjack_device as device:
    # all commands in this context block are executed as a transaction that cannot be interrupted
    current_ain0: float = device.ain0
    current_ain1: float = device.ain1
    current_ain2: float = device.ain2
```

Examples

Digital Input

To read the current state of `EIO0`, run:

```
from htf.labjack import T7

device = T7(host="my-labjack-device")
eio0: int = device.eio0
print(f"{eio0=}")
```

Digital Output

To set the current state of `EIO0` to 1, run:

```
from htf.labjack import T7

device = T7(host="my-labjack-device")
device.eio0 = 1
```

Analog Input

To read the current value of `AIN0`, run:

```
from htf.labjack import T7

device = T7(host="my-labjack-device")
device.setup_ain0(range_=5.0, settling_time_us=0, resolution_index=0)  # optional
ain0: float = device.ain0
print(f"{ain0=}")
```

Analog Output

To set `DAC0` to `5 V`, run:

```
from htf.labjack import T7

device = T7(host="my-labjack-device")
device.dac0 = 5.0
```

Example usage in tests

The following example shows a self test for LabJack T7 in a test with a fixture and settings.

Local settings are located in `settings.yml`:

```
labjack:
  hostname: 10.0.0.83
```

A fixture to use the device is located in `fixtures.py`:

```
import htf
from htf.labjack.t7 import T7


@htf.fixture(scope="test")
def t7(settings: htf.Settings) -> T7:
    device = T7(host=settings.labjack.hostname)
    yield device
    device.close()
```

An example test can be found in `example.py`:

```
import htf


def test_t7_analog_io(t7, step) -> None:
    """
    External wiring:
    DAC0<--->AIN0+AIN1
    DAC1<--->AIN2+AIN3
    """
    with step("Test analog IO"):
        with step("Setup AIN"):
            t7.setup_ain_all(range_=5.0, settling_time_us=10000)
            t7.setup_ain0(range_=5.0, settling_time_us=10000)
            t7.setup_ain1(range_=5.0, settling_time_us=10000)
            t7.setup_ain2(range_=5.0, settling_time_us=10000)
            t7.setup_ain3(range_=5.0, settling_time_us=10000)

        with step("Set all DAC to 0 V"):
            t7.dac0 = 0.0
            t7.dac1 = 0.0
            htf.assert_almost_equal(t7.ain0, 0.0, places=1)
            htf.assert_almost_equal(t7.ain1, 0.0, places=1)
            htf.assert_almost_equal(t7.ain2, 0.0, places=1)
            htf.assert_almost_equal(t7.ain3, 0.0, places=1)

        with step("Set DAC0 to 5 V"):
            t7.dac0 = 5.0
            htf.assert_almost_equal(t7.ain0, 5.0, places=1)
            htf.assert_almost_equal(t7.ain1, 5.0, places=1)
            htf.assert_almost_equal(t7.ain2, 0.0, places=1)
            htf.assert_almost_equal(t7.ain3, 0.0, places=1)

        with step("Set DAC1 to 3 V"):
            t7.dac1 = 3.0
            htf.assert_almost_equal(t7.ain0, 5.0, places=1)
            htf.assert_almost_equal(t7.ain1, 5.0, places=1)
            htf.assert_almost_equal(t7.ain2, 3.0, places=1)
```

```
            htf.assert_almost_equal(t7.ain3, 3.0, places=1)


def test_t7_digital_io(t7, step) -> None:
    """
    External wiring:

    FIO0<--->FIO1
    FIO2<--->GND
    FIO3<--->NC
    """
    with step("Test digital IO"):
        with step("Set FIO0 and FIO1 to input"):
            t7.fio0  # read value to make FIO0 an input
            t7.fio1  # read value to make FIO1 an input

        with step("Check that FIO0 and FIO1 both inputs"):
            print(f"{t7.fio0=}")
            print(f"{t7.fio1=}")
            htf.assert_equal(t7.fio0, 1)
            htf.assert_equal(t7.fio1, 1)

        with step("Check that FIO0 drives HIGH and FIO1 reads HIGH"):
            t7.fio0 = 1
            htf.assert_equal(t7.fio0, 1)
            htf.assert_equal(t7.fio1, 1)

        with step("Set FIO0 and FIO1 to input"):
            t7.fio0  # read value to make FIO0 an input
            t7.fio1  # read value to make FIO1 an input

        with step("Check that FIO1 drives HIGH and FIO0 reads HIGH"):
            t7.fio1 = 1
            htf.assert_equal(t7.fio1, 1)
            htf.assert_equal(t7.fio0, 1)

        with step("Set FIO0 and FIO1 to input"):
            t7.fio0  # read value to make FIO0 an input
            t7.fio1  # read value to make FIO1 an input

        with step("Check that FIO2 reads GND"):
            htf.assert_equal(t7.fio2, 0)

        with step("Check that FIO3 reads HIGH"):
            htf.assert_equal(t7.fio3, 1)


if __name__ == "__main__":
    htf.main()
```

LabJack T4




```
class htf.labjack.T4(*, name: str | None = None, host: str | None = None, port: int = 502,
debuglevel: int = 0, timeout: float = 1.0, retries: int = 0)
```

LabJack adapter for the T4.

Parameters:

- **name** – If name is set the USB device's name is checked against it. If host is None a USB device is opened.
- **host** – If host is set a Modbus-TCP connection is opened. Name must not be set at the same time.
- **port** – The Modbus-TCP port to be used
- **debuglevel** – If set the Modbus packets are printed
- **timeout** – The timeout for every request
- **retries** – The number of retries for Modbus communication

```
property ain0: float
```

Returns the voltage of the specified analog input.

Register: AIN0 (AIN#(0:11)@0)

Address: 0

```
property ain0_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN0_BINARY (AIN#(0:254)_BINARY@0)

Address: 50000

```
property ain0_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@0)

Address: 9300

```
property ain0_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@0)

Address: 9600

```
property ain0_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@0)

Address: 9900

```
property ain0_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@0)

Address: 10200

```
property ain0_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@0)

Address: 10500

```
property ain0_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@0)

Address: 10800

```
property ain0_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@0)

Address: 11100

```
property ain0_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@0)

Address: 11400

```
property ain0_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@0)

Address: 11700

```
property ain0_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@0)

Address: 12000

```
property ain0_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T;

25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN0_EF_INDEX (AIN#(0:149)_EF_INDEX@0)

Address: 9000

```
property ain0_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN0_EF_READ_A (AIN#(0:149)_EF_READ_A@0)

Address: 7000

```
property ain0_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN0_EF_READ_B (AIN#(0:149)_EF_READ_B@0)

Address: 7300

```
property ain0_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN0_EF_READ_C (AIN#(0:149)_EF_READ_C@0)

Address: 7600

```
property ain0_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN0_EF_READ_D (AIN#(0:149)_EF_READ_D@0)

Address: 7900

```
property ain0_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN0_NEGATIVE_CH (AIN#(0:3)_NEGATIVE_CH@0)

Address: 41000

```
property ain0_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN0_RANGE (AIN#(0:3)_RANGE@0)

Address: 40000

```
property ain0_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN0_RESOLUTION_INDEX (AIN#(0:3)_RESOLUTION_INDEX@0)

Address: 41500

```
property ain0_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN0_SETTLING_US (AIN#(0:3)_SETTLING_US@0)

Address: 42000

```
property ain1: float
```

Returns the voltage of the specified analog input.

Register: AIN1 (AIN#(0:11)@1)

Address: 2

```
property ain10: float
```

Returns the voltage of the specified analog input.

Register: AIN10 (AIN#(0:11)@10)

Address: 20

```
property ain100_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN100_BINARY (AIN#(0:254)_BINARY@100)

Address: 50200

```
property ain100_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@100)

Address: 9500

```
property ain100_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@100)

Address: 9800

```
property ain100_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@100)

Address: 10100

```
property ain100_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@100)

Address: 10400

```
property ain100_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@100)

Address: 10700

```
property ain100_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@100)

Address: 11000

```
property ain100_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@100)

Address: 11300

```
property ain100_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@100)

Address: 11600

```
property ain100_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@100)

Address: 11900

```
property ain100_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@100)

Address: 12200

```
property ain100_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN100_EF_INDEX (AIN#(0:149)_EF_INDEX@100)

Address: 9200

```
property ain100_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN100_EF_READ_A (AIN#(0:149)_EF_READ_A@100)

Address: 7200

```
property ain100_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN100_EF_READ_B (AIN#(0:149)_EF_READ_B@100)

Address: 7500

```
property ain100_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN100_EF_READ_C (AIN#(0:149)_EF_READ_C@100)

Address: 7800

```
property ain100_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN100_EF_READ_D (AIN#(0:149)_EF_READ_D@100)

Address: 8100

```
property ain101_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN101_BINARY (AIN#(0:254)_BINARY@101)

Address: 50202

```
property ain101_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@101)

Address: 9502

```
property ain101_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@101)

Address: 9802

```
property ain101_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@101)

Address: 10102

```
property ain101_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@101)

Address: 10402

```
property ain101_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@101)

Address: 10702

```
property ain101_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@101)

Address: 11002

```
property ain101_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@101)

Address: 11302

```
property ain101_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@101)

Address: 11602

```
property ain101_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@101)

Address: 11902

```
property ain101_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@101)

Address: 12202

```
property ain101_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN101_EF_INDEX (AIN#(0:149)_EF_INDEX@101)

Address: 9202

```
property ain101_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN101_EF_READ_A (AIN#(0:149)_EF_READ_A@101)

Address: 7202

```
property ain101_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN101_EF_READ_B (AIN#(0:149)_EF_READ_B@101)

Address: 7502

```
property ain101_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN101_EF_READ_C (AIN#(0:149)_EF_READ_C@101)

Address: 7802

```
property ain101_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN101_EF_READ_D (AIN#(0:149)_EF_READ_D@101)

Address: 8102

```
property ain102_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN102_BINARY (AIN#(0:254)_BINARY@102)

Address: 50204

```
property ain102_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@102)

Address: 9504

```
property ain102_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@102)

Address: 9804

```
property ain102_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@102)

Address: 10104

```
property ain102_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@102)

Address: 10404

```
property ain102_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@102)

Address: 10704

```
property ain102_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@102)

Address: 11004

```
property ain102_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@102)

Address: 11304

```
property ain102_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@102)

Address: 11604

```
property ain102_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@102)

Address: 11904

```
property ain102_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@102)

Address: 12204

```
property ain102_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN102_EF_INDEX (AIN#(0:149)_EF_INDEX@102)

Address: 9204

```
property ain102_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN102_EF_READ_A (AIN#(0:149)_EF_READ_A@102)

Address: 7204

```
property ain102_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN102_EF_READ_B (AIN#(0:149)_EF_READ_B@102)

Address: 7504

```
property ain102_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN102_EF_READ_C (AIN#(0:149)_EF_READ_C@102)

Address: 7804

```
property ain102_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN102_EF_READ_D (AIN#(0:149)_EF_READ_D@102)

Address: 8104

```
property ain103_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN103_BINARY (AIN#(0:254)_BINARY@103)

Address: 50206

```
property ain103_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@103)

Address: 9506

```
property ain103_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@103)

Address: 9806

```
property ain103_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@103)

Address: 10106

```
property ain103_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@103)

Address: 10406

```
property ain103_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@103)

Address: 10706

```
property ain103_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@103)

Address: 11006

```
property ain103_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@103)

Address: 11306

```
property ain103_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@103)

Address: 11606

```
property ain103_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@103)

Address: 11906

```
property ain103_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@103)

Address: 12206

```
property ain103_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN103_EF_INDEX (AIN#(0:149)_EF_INDEX@103)

Address: 9206

```
property ain103_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN103_EF_READ_A (AIN#(0:149)_EF_READ_A@103)

Address: 7206

```
property ain103_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN103_EF_READ_B (AIN#(0:149)_EF_READ_B@103)

Address: 7506

```
property ain103_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN103_EF_READ_C (AIN#(0:149)_EF_READ_C@103)

Address: 7806

```
property ain103_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN103_EF_READ_D (AIN#(0:149)_EF_READ_D@103)

Address: 8106

```
property ain104_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN104_BINARY (AIN#(0:254)_BINARY@104)

Address: 50208

```
property ain104_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@104)

Address: 9508

```
property ain104_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@104)

Address: 9808

```
property ain104_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@104)

Address: 10108

```
property ain104_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@104)

Address: 10408

```
property ain104_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@104)

Address: 10708

```
property ain104_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@104)

Address: 11008

```
property ain104_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@104)

Address: 11308

```
property ain104_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@104)

Address: 11608

```
property ain104_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@104)

Address: 11908

```
property ain104_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@104)

Address: 12208

```
property ain104_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN104_EF_INDEX (AIN#(0:149)_EF_INDEX@104)

Address: 9208

```
property ain104_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN104_EF_READ_A (AIN#(0:149)_EF_READ_A@104)

Address: 7208

```
property ain104_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN104_EF_READ_B (AIN#(0:149)_EF_READ_B@104)

Address: 7508

```
property ain104_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN104_EF_READ_C (AIN#(0:149)_EF_READ_C@104)

Address: 7808

```
property ain104_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN104_EF_READ_D (AIN#(0:149)_EF_READ_D@104)

Address: 8108

```
property ain105_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN105_BINARY (AIN#(0:254)_BINARY@105)

Address: 50210

```
property ain105_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@105)

Address: 9510

```
property ain105_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@105)

Address: 9810

```
property ain105_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@105)

Address: 10110

```
property ain105_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@105)

Address: 10410

```
property ain105_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@105)

Address: 10710

```
property ain105_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@105)

Address: 11010

```
property ain105_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@105)

Address: 11310

```
property ain105_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@105)

Address: 11610

```
property ain105_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@105)

Address: 11910

```
property ain105_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@105)

Address: 12210

```
property ain105_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN105_EF_INDEX (AIN#(0:149)_EF_INDEX@105)

Address: 9210

```
property ain105_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN105_EF_READ_A (AIN#(0:149)_EF_READ_A@105)

Address: 7210

```
property ain105_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN105_EF_READ_B (AIN#(0:149)_EF_READ_B@105)

Address: 7510

```
property ain105_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN105_EF_READ_C (AIN#(0:149)_EF_READ_C@105)

Address: 7810

```
property ain105_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN105_EF_READ_D (AIN#(0:149)_EF_READ_D@105)

Address: 8110

```
property ain106_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN106_BINARY (AIN#(0:254)_BINARY@106)

Address: 50212

```
property ain106_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@106)

Address: 9512

```
property ain106_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@106)

Address: 9812

```
property ain106_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@106)

Address: 10112

```
property ain106_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@106)

Address: 10412

```
property ain106_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@106)

Address: 10712

```
property ain106_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@106)

Address: 11012

```
property ain106_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@106)

Address: 11312

```
property ain106_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@106)

Address: 11612

```
property ain106_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@106)

Address: 11912

```
property ain106_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@106)

Address: 12212

```
property ain106_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN106_EF_INDEX (AIN#(0:149)_EF_INDEX@106)

Address: 9212

```
property ain106_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN106_EF_READ_A (AIN#(0:149)_EF_READ_A@106)

Address: 7212

```
property ain106_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN106_EF_READ_B (AIN#(0:149)_EF_READ_B@106)

Address: 7512

```
property ain106_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN106_EF_READ_C (AIN#(0:149)_EF_READ_C@106)

Address: 7812

```
property ain106_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN106_EF_READ_D (AIN#(0:149)_EF_READ_D@106)

Address: 8112

```
property ain107_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN107_BINARY (AIN#(0:254)_BINARY@107)

Address: 50214

```
property ain107_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@107)

Address: 9514

```
property ain107_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@107)

Address: 9814

```
property ain107_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@107)

Address: 10114

```
property ain107_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@107)

Address: 10414

```
property ain107_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@107)

Address: 10714

```
property ain107_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@107)

Address: 11014

```
property ain107_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@107)

Address: 11314

```
property ain107_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@107)

Address: 11614

```
property ain107_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@107)

Address: 11914

```
property ain107_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@107)

Address: 12214

```
property ain107_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN107_EF_INDEX (AIN#(0:149)_EF_INDEX@107)

Address: 9214

```
property ain107_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN107_EF_READ_A (AIN#(0:149)_EF_READ_A@107)

Address: 7214

```
property ain107_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN107_EF_READ_B (AIN#(0:149)_EF_READ_B@107)

Address: 7514

```
property ain107_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN107_EF_READ_C (AIN#(0:149)_EF_READ_C@107)

Address: 7814

```
property ain107_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN107_EF_READ_D (AIN#(0:149)_EF_READ_D@107)

Address: 8114

```
property ain108_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN108_BINARY (AIN#(0:254)_BINARY@108)

Address: 50216

```
property ain108_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@108)

Address: 9516

```
property ain108_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@108)

Address: 9816

```
property ain108_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@108)

Address: 10116

```
property ain108_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@108)

Address: 10416

```
property ain108_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@108)

Address: 10716

```
property ain108_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@108)

Address: 11016

```
property ain108_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@108)

Address: 11316

```
property ain108_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@108)

Address: 11616

```
property ain108_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@108)

Address: 11916

```
property ain108_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@108)

Address: 12216

```
property ain108_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN108_EF_INDEX (AIN#(0:149)_EF_INDEX@108)

Address: 9216

```
property ain108_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN108_EF_READ_A (AIN#(0:149)_EF_READ_A@108)

Address: 7216

```
property ain108_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN108_EF_READ_B (AIN#(0:149)_EF_READ_B@108)

Address: 7516

```
property ain108_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN108_EF_READ_C (AIN#(0:149)_EF_READ_C@108)

Address: 7816

```
property ain108_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN108_EF_READ_D (AIN#(0:149)_EF_READ_D@108)

Address: 8116

```
property ain109_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN109_BINARY (AIN#(0:254)_BINARY@109)

Address: 50218

```
property ain109_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@109)

Address: 9518

```
property ain109_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@109)

Address: 9818

```
property ain109_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@109)

Address: 10118

```
property ain109_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@109)

Address: 10418

```
property ain109_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@109)

Address: 10718

```
property ain109_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@109)

Address: 11018

```
property ain109_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@109)

Address: 11318

```
property ain109_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@109)

Address: 11618

```
property ain109_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@109)

Address: 11918

```
property ain109_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@109)

Address: 12218

```
property ain109_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN109_EF_INDEX (AIN#(0:149)_EF_INDEX@109)

Address: 9218

```
property ain109_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN109_EF_READ_A (AIN#(0:149)_EF_READ_A@109)

Address: 7218

```
property ain109_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN109_EF_READ_B (AIN#(0:149)_EF_READ_B@109)

Address: 7518

```
property ain109_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN109_EF_READ_C (AIN#(0:149)_EF_READ_C@109)

Address: 7818

```
property ain109_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN109_EF_READ_D (AIN#(0:149)_EF_READ_D@109)

Address: 8118

```
property ain10_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN10_BINARY (AIN#(0:254)_BINARY@10)

Address: 50020

```
property ain10_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@10)

Address: 9320

```
property ain10_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@10)

Address: 9620

```
property ain10_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@10)

Address: 9920

```
property ain10_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@10)

Address: 10220

```
property ain10_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@10)

Address: 10520

```
property ain10_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@10)

Address: 10820

```
property ain10_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@10)

Address: 11120

```
property ain10_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@10)

Address: 11420

```
property ain10_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@10)

Address: 11720

```
property ain10_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@10)

Address: 12020

```
property ain10_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN10_EF_INDEX (AIN#(0:149)_EF_INDEX@10)

Address: 9020

```
property ain10_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN10_EF_READ_A (AIN#(0:149)_EF_READ_A@10)

Address: 7020

```
property ain10_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN10_EF_READ_B (AIN#(0:149)_EF_READ_B@10)

Address: 7320

```
property ain10_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN10_EF_READ_C (AIN#(0:149)_EF_READ_C@10)

Address: 7620

```
property ain10_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN10_EF_READ_D (AIN#(0:149)_EF_READ_D@10)

Address: 7920

```
property ain11: float
```

Returns the voltage of the specified analog input.

Register: AIN11 (AIN#(0:11)@11)

Address: 22

```
property ain110_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN110_BINARY (AIN#(0:254)_BINARY@110)

Address: 50220

```
property ain110_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@110)

Address: 9520

```
property ain110_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@110)

Address: 9820

```
property ain110_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@110)

Address: 10120

```
property ain110_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@110)

Address: 10420

```
property ain110_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@110)

Address: 10720

```
property ain110_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@110)

Address: 11020

```
property ain110_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@110)

Address: 11320

```
property ain110_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@110)

Address: 11620

```
property ain110_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@110)

Address: 11920

```
property ain110_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@110)

Address: 12220

```
property ain110_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN110_EF_INDEX (AIN#(0:149)_EF_INDEX@110)

Address: 9220

```
property ain110_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN110_EF_READ_A (AIN#(0:149)_EF_READ_A@110)

Address: 7220

```
property ain110_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN110_EF_READ_B (AIN#(0:149)_EF_READ_B@110)

Address: 7520

```
property ain110_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN110_EF_READ_C (AIN#(0:149)_EF_READ_C@110)

Address: 7820

```
property ain110_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN110_EF_READ_D (AIN#(0:149)_EF_READ_D@110)

Address: 8120

```
property ain111_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN111_BINARY (AIN#(0:254)_BINARY@111)

Address: 50222

```
property ain111_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@111)

Address: 9522

```
property ain111_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@111)

Address: 9822

```
property ain111_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@111)

Address: 10122

```
property ain111_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@111)

Address: 10422

```
property ain111_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@111)

Address: 10722

```
property ain111_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@111)

Address: 11022

```
property ain111_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@111)

Address: 11322

```
property ain111_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@111)

Address: 11622

```
property ain111_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@111)

Address: 11922

```
property ain111_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@111)

Address: 12222

```
property ain111_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN111_EF_INDEX (AIN#(0:149)_EF_INDEX@111)

Address: 9222

```
property ain111_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN111_EF_READ_A (AIN#(0:149)_EF_READ_A@111)

Address: 7222

```
property ain111_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN111_EF_READ_B (AIN#(0:149)_EF_READ_B@111)

Address: 7522

```
property ain111_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN111_EF_READ_C (AIN#(0:149)_EF_READ_C@111)

Address: 7822

```
property ain111_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN111_EF_READ_D (AIN#(0:149)_EF_READ_D@111)

Address: 8122

```
property ain112_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN112_BINARY (AIN#(0:254)_BINARY@112)

Address: 50224

```
property ain112_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@112)

Address: 9524

```
property ain112_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@112)

Address: 9824

```
property ain112_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@112)

Address: 10124

```
property ain112_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@112)

Address: 10424

```
property ain112_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@112)

Address: 10724

```
property ain112_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@112)

Address: 11024

```
property ain112_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@112)

Address: 11324

```
property ain112_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@112)

Address: 11624

```
property ain112_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@112)

Address: 11924

```
property ain112_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@112)

Address: 12224

```
property ain112_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN112_EF_INDEX (AIN#(0:149)_EF_INDEX@112)

Address: 9224

```
property ain112_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN112_EF_READ_A (AIN#(0:149)_EF_READ_A@112)

Address: 7224

```
property ain112_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN112_EF_READ_B (AIN#(0:149)_EF_READ_B@112)

Address: 7524

```
property ain112_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN112_EF_READ_C (AIN#(0:149)_EF_READ_C@112)

Address: 7824

```
property ain112_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN112_EF_READ_D (AIN#(0:149)_EF_READ_D@112)

Address: 8124

```
property ain113_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN113_BINARY (AIN#(0:254)_BINARY@113)

Address: 50226

```
property ain113_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@113)

Address: 9526

```
property ain113_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@113)

Address: 9826

```
property ain113_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@113)

Address: 10126

```
property ain113_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@113)

Address: 10426

```
property ain113_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@113)

Address: 10726

```
property ain113_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@113)

Address: 11026

```
property ain113_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@113)

Address: 11326

```
property ain113_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@113)

Address: 11626

```
property ain113_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@113)

Address: 11926

```
property ain113_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@113)

Address: 12226

```
property ain113_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN113_EF_INDEX (AIN#(0:149)_EF_INDEX@113)

Address: 9226

```
property ain113_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN113_EF_READ_A (AIN#(0:149)_EF_READ_A@113)

Address: 7226

```
property ain113_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN113_EF_READ_B (AIN#(0:149)_EF_READ_B@113)

Address: 7526

```
property ain113_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN113_EF_READ_C (AIN#(0:149)_EF_READ_C@113)

Address: 7826

```
property ain113_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN113_EF_READ_D (AIN#(0:149)_EF_READ_D@113)

Address: 8126

```
property ain114_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN114_BINARY (AIN#(0:254)_BINARY@114)

Address: 50228

```
property ain114_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@114)

Address: 9528

```
property ain114_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@114)

Address: 9828

```
property ain114_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@114)

Address: 10128

```
property ain114_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@114)

Address: 10428

```
property ain114_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@114)

Address: 10728

```
property ain114_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@114)

Address: 11028

```
property ain114_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@114)

Address: 11328

```
property ain114_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@114)

Address: 11628

```
property ain114_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@114)

Address: 11928

```
property ain114_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@114)

Address: 12228

```
property ain114_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN114_EF_INDEX (AIN#(0:149)_EF_INDEX@114)

Address: 9228

```
property ain114_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN114_EF_READ_A (AIN#(0:149)_EF_READ_A@114)

Address: 7228

```
property ain114_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN114_EF_READ_B (AIN#(0:149)_EF_READ_B@114)

Address: 7528

```
property ain114_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN114_EF_READ_C (AIN#(0:149)_EF_READ_C@114)

Address: 7828

```
property ain114_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN114_EF_READ_D (AIN#(0:149)_EF_READ_D@114)

Address: 8128

```
property ain115_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN115_BINARY (AIN#(0:254)_BINARY@115)

Address: 50230

```
property ain115_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@115)

Address: 9530

```
property ain115_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@115)

Address: 9830

```
property ain115_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@115)

Address: 10130

```
property ain115_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@115)

Address: 10430

```
property ain115_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@115)

Address: 10730

```
property ain115_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@115)

Address: 11030

```
property ain115_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@115)

Address: 11330

```
property ain115_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@115)

Address: 11630

```
property ain115_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@115)

Address: 11930

```
property ain115_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@115)

Address: 12230

```
property ain115_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN115_EF_INDEX (AIN#(0:149)_EF_INDEX@115)

Address: 9230

```
property ain115_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN115_EF_READ_A (AIN#(0:149)_EF_READ_A@115)

Address: 7230

```
property ain115_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN115_EF_READ_B (AIN#(0:149)_EF_READ_B@115)

Address: 7530

```
property ain115_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN115_EF_READ_C (AIN#(0:149)_EF_READ_C@115)

Address: 7830

```
property ain115_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN115_EF_READ_D (AIN#(0:149)_EF_READ_D@115)

Address: 8130

```
property ain116_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN116_BINARY (AIN#(0:254)_BINARY@116)

Address: 50232

```
property ain116_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@116)

Address: 9532

```
property ain116_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@116)

Address: 9832

```
property ain116_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@116)

Address: 10132

```
property ain116_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@116)

Address: 10432

```
property ain116_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@116)

Address: 10732

```
property ain116_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@116)

Address: 11032

```
property ain116_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@116)

Address: 11332

```
property ain116_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@116)

Address: 11632

```
property ain116_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@116)

Address: 11932

```
property ain116_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@116)

Address: 12232

```
property ain116_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN116_EF_INDEX (AIN#(0:149)_EF_INDEX@116)

Address: 9232

```
property ain116_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN116_EF_READ_A (AIN#(0:149)_EF_READ_A@116)

Address: 7232

```
property ain116_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN116_EF_READ_B (AIN#(0:149)_EF_READ_B@116)

Address: 7532

```
property ain116_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN116_EF_READ_C (AIN#(0:149)_EF_READ_C@116)

Address: 7832

```
property ain116_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN116_EF_READ_D (AIN#(0:149)_EF_READ_D@116)

Address: 8132

```
property ain117_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN117_BINARY (AIN#(0:254)_BINARY@117)

Address: 50234

```
property ain117_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@117)

Address: 9534

```
property ain117_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@117)

Address: 9834

```
property ain117_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@117)

Address: 10134

```
property ain117_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@117)

Address: 10434

```
property ain117_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@117)

Address: 10734

```
property ain117_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@117)

Address: 11034

```
property ain117_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@117)

Address: 11334

```
property ain117_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@117)

Address: 11634

```
property ain117_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@117)

Address: 11934

```
property ain117_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@117)

Address: 12234

```
property ain117_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN117_EF_INDEX (AIN#(0:149)_EF_INDEX@117)

Address: 9234

```
property ain117_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN117_EF_READ_A (AIN#(0:149)_EF_READ_A@117)

Address: 7234

```
property ain117_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN117_EF_READ_B (AIN#(0:149)_EF_READ_B@117)

Address: 7534

```
property ain117_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN117_EF_READ_C (AIN#(0:149)_EF_READ_C@117)

Address: 7834

```
property ain117_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN117_EF_READ_D (AIN#(0:149)_EF_READ_D@117)

Address: 8134

```
property ain118_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN118_BINARY (AIN#(0:254)_BINARY@118)

Address: 50236

```
property ain118_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@118)

Address: 9536

```
property ain118_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@118)

Address: 9836

```
property ain118_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@118)

Address: 10136

```
property ain118_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@118)

Address: 10436

```
property ain118_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@118)

Address: 10736

```
property ain118_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@118)

Address: 11036

```
property ain118_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@118)

Address: 11336

```
property ain118_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@118)

Address: 11636

```
property ain118_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@118)

Address: 11936

```
property ain118_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@118)

Address: 12236

```
property ain118_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN118_EF_INDEX (AIN#(0:149)_EF_INDEX@118)

Address: 9236

```
property ain118_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN118_EF_READ_A (AIN#(0:149)_EF_READ_A@118)

Address: 7236

```
property ain118_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN118_EF_READ_B (AIN#(0:149)_EF_READ_B@118)

Address: 7536

```
property ain118_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN118_EF_READ_C (AIN#(0:149)_EF_READ_C@118)

Address: 7836

```
property ain118_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN118_EF_READ_D (AIN#(0:149)_EF_READ_D@118)

Address: 8136

```
property ain119_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN119_BINARY (AIN#(0:254)_BINARY@119)

Address: 50238

```
property ain119_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@119)

Address: 9538

```
property ain119_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@119)

Address: 9838

```
property ain119_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@119)

Address: 10138

```
property ain119_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@119)

Address: 10438

```
property ain119_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@119)

Address: 10738

```
property ain119_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@119)

Address: 11038

```
property ain119_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@119)

Address: 11338

```
property ain119_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@119)

Address: 11638

```
property ain119_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@119)

Address: 11938

```
property ain119_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@119)

Address: 12238

```
property ain119_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN119_EF_INDEX (AIN#(0:149)_EF_INDEX@119)

Address: 9238

```
property ain119_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN119_EF_READ_A (AIN#(0:149)_EF_READ_A@119)

Address: 7238

```
property ain119_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN119_EF_READ_B (AIN#(0:149)_EF_READ_B@119)

Address: 7538

```
property ain119_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN119_EF_READ_C (AIN#(0:149)_EF_READ_C@119)

Address: 7838

```
property ain119_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN119_EF_READ_D (AIN#(0:149)_EF_READ_D@119)

Address: 8138

```
property ain11_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN11_BINARY (AIN#(0:254)_BINARY@11)

Address: 50022

```
property ain11_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@11)

Address: 9322

```
property ain11_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@11)

Address: 9622

```
property ain11_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@11)

Address: 9922

```
property ain11_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@11)

Address: 10222

```
property ain11_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@11)

Address: 10522

```
property ain11_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@11)

Address: 10822

```
property ain11_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@11)

Address: 11122

```
property ain11_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@11)

Address: 11422

```
property ain11_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@11)

Address: 11722

```
property ain11_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@11)

Address: 12022

```
property ain11_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN11_EF_INDEX (AIN#(0:149)_EF_INDEX@11)

Address: 9022

```
property ain11_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN11_EF_READ_A (AIN#(0:149)_EF_READ_A@11)

Address: 7022

```
property ain11_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN11_EF_READ_B (AIN#(0:149)_EF_READ_B@11)

Address: 7322

```
property ain11_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN11_EF_READ_C (AIN#(0:149)_EF_READ_C@11)

Address: 7622

```
property ain11_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN11_EF_READ_D (AIN#(0:149)_EF_READ_D@11)

Address: 7922

```
property ain120_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN120_BINARY (AIN#(0:254)_BINARY@120)

Address: 50240

```
property ain120_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@120)

Address: 9540

```
property ain120_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@120)

Address: 9840

```
property ain120_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@120)

Address: 10140

```
property ain120_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@120)

Address: 10440

```
property ain120_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@120)

Address: 10740

```
property ain120_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@120)

Address: 11040

```
property ain120_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@120)

Address: 11340

```
property ain120_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@120)

Address: 11640

```
property ain120_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@120)

Address: 11940

```
property ain120_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@120)

Address: 12240

```
property ain120_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN120_EF_INDEX (AIN#(0:149)_EF_INDEX@120)

Address: 9240

```
property ain120_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN120_EF_READ_A (AIN#(0:149)_EF_READ_A@120)

Address: 7240

```
property ain120_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN120_EF_READ_B (AIN#(0:149)_EF_READ_B@120)

Address: 7540

```
property ain120_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN120_EF_READ_C (AIN#(0:149)_EF_READ_C@120)

Address: 7840

```
property ain120_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN120_EF_READ_D (AIN#(0:149)_EF_READ_D@120)

Address: 8140

```
property ain121_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN121_BINARY (AIN#(0:254)_BINARY@121)

Address: 50242

```
property ain121_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@121)

Address: 9542

```
property ain121_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@121)

Address: 9842

```
property ain121_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@121)

Address: 10142

```
property ain121_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@121)

Address: 10442

```
property ain121_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@121)

Address: 10742

```
property ain121_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@121)

Address: 11042

```
property ain121_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@121)

Address: 11342

```
property ain121_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@121)

Address: 11642

```
property ain121_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@121)

Address: 11942

```
property ain121_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@121)

Address: 12242

```
property ain121_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN121_EF_INDEX (AIN#(0:149)_EF_INDEX@121)

Address: 9242

```
property ain121_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN121_EF_READ_A (AIN#(0:149)_EF_READ_A@121)

Address: 7242

```
property ain121_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN121_EF_READ_B (AIN#(0:149)_EF_READ_B@121)

Address: 7542

```
property ain121_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN121_EF_READ_C (AIN#(0:149)_EF_READ_C@121)

Address: 7842

```
property ain121_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN121_EF_READ_D (AIN#(0:149)_EF_READ_D@121)

Address: 8142

```
property ain122_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN122_BINARY (AIN#(0:254)_BINARY@122)

Address: 50244

```
property ain122_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@122)

Address: 9544

```
property ain122_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@122)

Address: 9844

```
property ain122_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@122)

Address: 10144

```
property ain122_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@122)

Address: 10444

```
property ain122_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@122)

Address: 10744

```
property ain122_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@122)

Address: 11044

```
property ain122_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@122)

Address: 11344

```
property ain122_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@122)

Address: 11644

```
property ain122_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@122)

Address: 11944

```
property ain122_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@122)

Address: 12244

```
property ain122_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN122_EF_INDEX (AIN#(0:149)_EF_INDEX@122)

Address: 9244

```
property ain122_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN122_EF_READ_A (AIN#(0:149)_EF_READ_A@122)

Address: 7244

```
property ain122_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN122_EF_READ_B (AIN#(0:149)_EF_READ_B@122)

Address: 7544

```
property ain122_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN122_EF_READ_C (AIN#(0:149)_EF_READ_C@122)

Address: 7844

```
property ain122_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN122_EF_READ_D (AIN#(0:149)_EF_READ_D@122)

Address: 8144

```
property ain123_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN123_BINARY (AIN#(0:254)_BINARY@123)

Address: 50246

```
property ain123_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@123)

Address: 9546

```
property ain123_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@123)

Address: 9846

```
property ain123_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@123)

Address: 10146

```
property ain123_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@123)

Address: 10446

```
property ain123_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@123)

Address: 10746

```
property ain123_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@123)

Address: 11046

```
property ain123_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@123)

Address: 11346

```
property ain123_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@123)

Address: 11646

```
property ain123_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@123)

Address: 11946

```
property ain123_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@123)

Address: 12246

```
property ain123_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN123_EF_INDEX (AIN#(0:149)_EF_INDEX@123)

Address: 9246

```
property ain123_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN123_EF_READ_A (AIN#(0:149)_EF_READ_A@123)

Address: 7246

```
property ain123_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN123_EF_READ_B (AIN#(0:149)_EF_READ_B@123)

Address: 7546

```
property ain123_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN123_EF_READ_C (AIN#(0:149)_EF_READ_C@123)

Address: 7846

```
property ain123_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN123_EF_READ_D (AIN#(0:149)_EF_READ_D@123)

Address: 8146

```
property ain124_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN124_BINARY (AIN#(0:254)_BINARY@124)

Address: 50248

```
property ain124_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@124)

Address: 9548

```
property ain124_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@124)

Address: 9848

```
property ain124_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@124)

Address: 10148

```
property ain124_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@124)

Address: 10448

```
property ain124_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@124)

Address: 10748

```
property ain124_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@124)

Address: 11048

```
property ain124_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@124)

Address: 11348

```
property ain124_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@124)

Address: 11648

```
property ain124_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@124)

Address: 11948

```
property ain124_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@124)

Address: 12248

```
property ain124_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN124_EF_INDEX (AIN#(0:149)_EF_INDEX@124)

Address: 9248

```
property ain124_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN124_EF_READ_A (AIN#(0:149)_EF_READ_A@124)

Address: 7248

```
property ain124_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN124_EF_READ_B (AIN#(0:149)_EF_READ_B@124)

Address: 7548

```
property ain124_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN124_EF_READ_C (AIN#(0:149)_EF_READ_C@124)

Address: 7848

```
property ain124_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN124_EF_READ_D (AIN#(0:149)_EF_READ_D@124)

Address: 8148

```
property ain125_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN125_BINARY (AIN#(0:254)_BINARY@125)

Address: 50250

```
property ain125_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@125)

Address: 9550

```
property ain125_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@125)

Address: 9850

```
property ain125_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@125)

Address: 10150

```
property ain125_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@125)

Address: 10450

```
property ain125_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@125)

Address: 10750

```
property ain125_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@125)

Address: 11050

```
property ain125_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@125)

Address: 11350

```
property ain125_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@125)

Address: 11650

```
property ain125_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@125)

Address: 11950

```
property ain125_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@125)

Address: 12250

```
property ain125_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN125_EF_INDEX (AIN#(0:149)_EF_INDEX@125)

Address: 9250

```
property ain125_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN125_EF_READ_A (AIN#(0:149)_EF_READ_A@125)

Address: 7250

```
property ain125_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN125_EF_READ_B (AIN#(0:149)_EF_READ_B@125)

Address: 7550

```
property ain125_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN125_EF_READ_C (AIN#(0:149)_EF_READ_C@125)

Address: 7850

```
property ain125_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN125_EF_READ_D (AIN#(0:149)_EF_READ_D@125)

Address: 8150

```
property ain126_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN126_BINARY (AIN#(0:254)_BINARY@126)

Address: 50252

```
property ain126_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@126)

Address: 9552

```
property ain126_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@126)

Address: 9852

```
property ain126_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@126)

Address: 10152

```
property ain126_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@126)

Address: 10452

```
property ain126_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@126)

Address: 10752

```
property ain126_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@126)

Address: 11052

```
property ain126_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@126)

Address: 11352

```
property ain126_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@126)

Address: 11652

```
property ain126_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@126)

Address: 11952

```
property ain126_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@126)

Address: 12252

```
property ain126_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN126_EF_INDEX (AIN#(0:149)_EF_INDEX@126)

Address: 9252

```
property ain126_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN126_EF_READ_A (AIN#(0:149)_EF_READ_A@126)

Address: 7252

```
property ain126_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN126_EF_READ_B (AIN#(0:149)_EF_READ_B@126)

Address: 7552

```
property ain126_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN126_EF_READ_C (AIN#(0:149)_EF_READ_C@126)

Address: 7852

```
property ain126_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN126_EF_READ_D (AIN#(0:149)_EF_READ_D@126)

Address: 8152

```
property ain127_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN127_BINARY (AIN#(0:254)_BINARY@127)

Address: 50254

```
property ain127_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@127)

Address: 9554

```
property ain127_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@127)

Address: 9854

```
property ain127_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@127)

Address: 10154

```
property ain127_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@127)

Address: 10454

```
property ain127_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@127)

Address: 10754

```
property ain127_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@127)

Address: 11054

```
property ain127_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@127)

Address: 11354

```
property ain127_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@127)

Address: 11654

```
property ain127_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@127)

Address: 11954

```
property ain127_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@127)

Address: 12254

```
property ain127_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN127_EF_INDEX (AIN#(0:149)_EF_INDEX@127)

Address: 9254

```
property ain127_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN127_EF_READ_A (AIN#(0:149)_EF_READ_A@127)

Address: 7254

```
property ain127_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN127_EF_READ_B (AIN#(0:149)_EF_READ_B@127)

Address: 7554

```
property ain127_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN127_EF_READ_C (AIN#(0:149)_EF_READ_C@127)

Address: 7854

```
property ain127_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN127_EF_READ_D (AIN#(0:149)_EF_READ_D@127)

Address: 8154

```
property ain128_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN128_BINARY (AIN#(0:254)_BINARY@128)

Address: 50256

```
property ain128_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@128)

Address: 9556

```
property ain128_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@128)

Address: 9856

```
property ain128_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@128)

Address: 10156

```
property ain128_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@128)

Address: 10456

```
property ain128_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@128)

Address: 10756

```
property ain128_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@128)

Address: 11056

```
property ain128_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@128)

Address: 11356

```
property ain128_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@128)

Address: 11656

```
property ain128_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@128)

Address: 11956

```
property ain128_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@128)

Address: 12256

```
property ain128_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN128_EF_INDEX (AIN#(0:149)_EF_INDEX@128)

Address: 9256

```
property ain128_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN128_EF_READ_A (AIN#(0:149)_EF_READ_A@128)

Address: 7256

```
property ain128_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN128_EF_READ_B (AIN#(0:149)_EF_READ_B@128)

Address: 7556

```
property ain128_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN128_EF_READ_C (AIN#(0:149)_EF_READ_C@128)

Address: 7856

```
property ain128_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN128_EF_READ_D (AIN#(0:149)_EF_READ_D@128)

Address: 8156

```
property ain129_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN129_BINARY (AIN#(0:254)_BINARY@129)

Address: 50258

```
property ain129_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@129)

Address: 9558

```
property ain129_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@129)

Address: 9858

```
property ain129_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@129)

Address: 10158

```
property ain129_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@129)

Address: 10458

```
property ain129_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@129)

Address: 10758

```
property ain129_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@129)

Address: 11058

```
property ain129_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@129)

Address: 11358

```
property ain129_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@129)

Address: 11658

```
property ain129_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@129)

Address: 11958

```
property ain129_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@129)

Address: 12258

```
property ain129_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN129_EF_INDEX (AIN#(0:149)_EF_INDEX@129)

Address: 9258

```
property ain129_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN129_EF_READ_A (AIN#(0:149)_EF_READ_A@129)

Address: 7258

```
property ain129_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN129_EF_READ_B (AIN#(0:149)_EF_READ_B@129)

Address: 7558

```
property ain129_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN129_EF_READ_C (AIN#(0:149)_EF_READ_C@129)

Address: 7858

```
property ain129_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN129_EF_READ_D (AIN#(0:149)_EF_READ_D@129)

Address: 8158

```
property ain12_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN12_BINARY (AIN#(0:254)_BINARY@12)

Address: 50024

```
property ain12_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@12)

Address: 9324

```
property ain12_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@12)

Address: 9624

```
property ain12_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@12)

Address: 9924

```
property ain12_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@12)

Address: 10224

```
property ain12_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@12)

Address: 10524

```
property ain12_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@12)

Address: 10824

```
property ain12_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@12)

Address: 11124

```
property ain12_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@12)

Address: 11424

```
property ain12_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@12)

Address: 11724

```
property ain12_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@12)

Address: 12024

```
property ain12_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN12_EF_INDEX (AIN#(0:149)_EF_INDEX@12)

Address: 9024

```
property ain12_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN12_EF_READ_A (AIN#(0:149)_EF_READ_A@12)

Address: 7024

```
property ain12_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN12_EF_READ_B (AIN#(0:149)_EF_READ_B@12)

Address: 7324

```
property ain12_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN12_EF_READ_C (AIN#(0:149)_EF_READ_C@12)

Address: 7624

```
property ain12_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN12_EF_READ_D (AIN#(0:149)_EF_READ_D@12)

Address: 7924

```
property ain130_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN130_BINARY (AIN#(0:254)_BINARY@130)

Address: 50260

```
property ain130_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@130)

Address: 9560

```
property ain130_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@130)

Address: 9860

```
property ain130_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@130)

Address: 10160

```
property ain130_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@130)

Address: 10460

```
property ain130_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@130)

Address: 10760

```
property ain130_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@130)

Address: 11060

```
property ain130_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@130)

Address: 11360

```
property ain130_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@130)

Address: 11660

```
property ain130_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@130)

Address: 11960

```
property ain130_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@130)

Address: 12260

```
property ain130_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN130_EF_INDEX (AIN#(0:149)_EF_INDEX@130)

Address: 9260

```
property ain130_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN130_EF_READ_A (AIN#(0:149)_EF_READ_A@130)

Address: 7260

```
property ain130_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN130_EF_READ_B (AIN#(0:149)_EF_READ_B@130)

Address: 7560

```
property ain130_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN130_EF_READ_C (AIN#(0:149)_EF_READ_C@130)

Address: 7860

```
property ain130_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN130_EF_READ_D (AIN#(0:149)_EF_READ_D@130)

Address: 8160

```
property ain131_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN131_BINARY (AIN#(0:254)_BINARY@131)

Address: 50262

```
property ain131_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@131)

Address: 9562

```
property ain131_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@131)

Address: 9862

```
property ain131_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@131)

Address: 10162

```
property ain131_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@131)

Address: 10462

```
property ain131_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@131)

Address: 10762

```
property ain131_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@131)

Address: 11062

```
property ain131_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@131)

Address: 11362

```
property ain131_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@131)

Address: 11662

```
property ain131_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@131)

Address: 11962

```
property ain131_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@131)

Address: 12262

```
property ain131_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN131_EF_INDEX (AIN#(0:149)_EF_INDEX@131)

Address: 9262

```
property ain131_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN131_EF_READ_A (AIN#(0:149)_EF_READ_A@131)

Address: 7262

```
property ain131_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN131_EF_READ_B (AIN#(0:149)_EF_READ_B@131)

Address: 7562

```
property ain131_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN131_EF_READ_C (AIN#(0:149)_EF_READ_C@131)

Address: 7862

```
property ain131_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN131_EF_READ_D (AIN#(0:149)_EF_READ_D@131)

Address: 8162

```
property ain132_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN132_BINARY (AIN#(0:254)_BINARY@132)

Address: 50264

```
property ain132_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@132)

Address: 9564

```
property ain132_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@132)

Address: 9864

```
property ain132_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@132)

Address: 10164

```
property ain132_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@132)

Address: 10464

```
property ain132_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@132)

Address: 10764

```
property ain132_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@132)

Address: 11064

```
property ain132_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@132)

Address: 11364

```
property ain132_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@132)

Address: 11664

```
property ain132_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@132)

Address: 11964

```
property ain132_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@132)

Address: 12264

```
property ain132_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN132_EF_INDEX (AIN#(0:149)_EF_INDEX@132)

Address: 9264

```
property ain132_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN132_EF_READ_A (AIN#(0:149)_EF_READ_A@132)

Address: 7264

```
property ain132_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN132_EF_READ_B (AIN#(0:149)_EF_READ_B@132)

Address: 7564

```
property ain132_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN132_EF_READ_C (AIN#(0:149)_EF_READ_C@132)

Address: 7864

```
property ain132_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN132_EF_READ_D (AIN#(0:149)_EF_READ_D@132)

Address: 8164

```
property ain133_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN133_BINARY (AIN#(0:254)_BINARY@133)

Address: 50266

```
property ain133_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@133)

Address: 9566

```
property ain133_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@133)

Address: 9866

```
property ain133_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@133)

Address: 10166

```
property ain133_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@133)

Address: 10466

```
property ain133_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@133)

Address: 10766

```
property ain133_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@133)

Address: 11066

```
property ain133_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@133)

Address: 11366

```
property ain133_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@133)

Address: 11666

```
property ain133_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@133)

Address: 11966

```
property ain133_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@133)

Address: 12266

```
property ain133_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN133_EF_INDEX (AIN#(0:149)_EF_INDEX@133)

Address: 9266

```
property ain133_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN133_EF_READ_A (AIN#(0:149)_EF_READ_A@133)

Address: 7266

```
property ain133_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN133_EF_READ_B (AIN#(0:149)_EF_READ_B@133)

Address: 7566

```
property ain133_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN133_EF_READ_C (AIN#(0:149)_EF_READ_C@133)

Address: 7866

```
property ain133_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN133_EF_READ_D (AIN#(0:149)_EF_READ_D@133)

Address: 8166

```
property ain134_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN134_BINARY (AIN#(0:254)_BINARY@134)

Address: 50268

```
property ain134_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@134)

Address: 9568

```
property ain134_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@134)

Address: 9868

```
property ain134_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@134)

Address: 10168

```
property ain134_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@134)

Address: 10468

```
property ain134_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@134)

Address: 10768

```
property ain134_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@134)

Address: 11068

```
property ain134_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@134)

Address: 11368

```
property ain134_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@134)

Address: 11668

```
property ain134_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@134)

Address: 11968

```
property ain134_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@134)

Address: 12268

```
property ain134_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN134_EF_INDEX (AIN#(0:149)_EF_INDEX@134)

Address: 9268

```
property ain134_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN134_EF_READ_A (AIN#(0:149)_EF_READ_A@134)

Address: 7268

```
property ain134_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN134_EF_READ_B (AIN#(0:149)_EF_READ_B@134)

Address: 7568

```
property ain134_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN134_EF_READ_C (AIN#(0:149)_EF_READ_C@134)

Address: 7868

```
property ain134_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN134_EF_READ_D (AIN#(0:149)_EF_READ_D@134)

Address: 8168

```
property ain135_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN135_BINARY (AIN#(0:254)_BINARY@135)

Address: 50270

```
property ain135_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@135)

Address: 9570

```
property ain135_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@135)

Address: 9870

```
property ain135_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@135)

Address: 10170

```
property ain135_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@135)

Address: 10470

```
property ain135_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@135)

Address: 10770

```
property ain135_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@135)

Address: 11070

```
property ain135_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@135)

Address: 11370

```
property ain135_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@135)

Address: 11670

```
property ain135_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@135)

Address: 11970

```
property ain135_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@135)

Address: 12270

```
property ain135_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN135_EF_INDEX (AIN#(0:149)_EF_INDEX@135)

Address: 9270

```
property ain135_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN135_EF_READ_A (AIN#(0:149)_EF_READ_A@135)

Address: 7270

```
property ain135_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN135_EF_READ_B (AIN#(0:149)_EF_READ_B@135)

Address: 7570

```
property ain135_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN135_EF_READ_C (AIN#(0:149)_EF_READ_C@135)

Address: 7870

```
property ain135_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN135_EF_READ_D (AIN#(0:149)_EF_READ_D@135)

Address: 8170

```
property ain136_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN136_BINARY (AIN#(0:254)_BINARY@136)

Address: 50272

```
property ain136_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@136)

Address: 9572

```
property ain136_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@136)

Address: 9872

```
property ain136_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@136)

Address: 10172

```
property ain136_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@136)

Address: 10472

```
property ain136_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@136)

Address: 10772

```
property ain136_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@136)

Address: 11072

```
property ain136_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@136)

Address: 11372

```
property ain136_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@136)

Address: 11672

```
property ain136_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@136)

Address: 11972

```
property ain136_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@136)

Address: 12272

```
property ain136_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN136_EF_INDEX (AIN#(0:149)_EF_INDEX@136)

Address: 9272

```
property ain136_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN136_EF_READ_A (AIN#(0:149)_EF_READ_A@136)

Address: 7272

```
property ain136_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN136_EF_READ_B (AIN#(0:149)_EF_READ_B@136)

Address: 7572

```
property ain136_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN136_EF_READ_C (AIN#(0:149)_EF_READ_C@136)

Address: 7872

```
property ain136_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN136_EF_READ_D (AIN#(0:149)_EF_READ_D@136)

Address: 8172

```
property ain137_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN137_BINARY (AIN#(0:254)_BINARY@137)

Address: 50274

```
property ain137_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@137)

Address: 9574

```
property ain137_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@137)

Address: 9874

```
property ain137_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@137)

Address: 10174

```
property ain137_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@137)

Address: 10474

```
property ain137_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@137)

Address: 10774

```
property ain137_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@137)

Address: 11074

```
property ain137_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@137)

Address: 11374

```
property ain137_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@137)

Address: 11674

```
property ain137_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@137)

Address: 11974

```
property ain137_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@137)

Address: 12274

```
property ain137_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN137_EF_INDEX (AIN#(0:149)_EF_INDEX@137)

Address: 9274

```
property ain137_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN137_EF_READ_A (AIN#(0:149)_EF_READ_A@137)

Address: 7274

```
property ain137_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN137_EF_READ_B (AIN#(0:149)_EF_READ_B@137)

Address: 7574

```
property ain137_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN137_EF_READ_C (AIN#(0:149)_EF_READ_C@137)

Address: 7874

```
property ain137_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN137_EF_READ_D (AIN#(0:149)_EF_READ_D@137)

Address: 8174

```
property ain138_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN138_BINARY (AIN#(0:254)_BINARY@138)

Address: 50276

```
property ain138_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@138)

Address: 9576

```
property ain138_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@138)

Address: 9876

```
property ain138_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@138)

Address: 10176

```
property ain138_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@138)

Address: 10476

```
property ain138_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@138)

Address: 10776

```
property ain138_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@138)

Address: 11076

```
property ain138_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@138)

Address: 11376

```
property ain138_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@138)

Address: 11676

```
property ain138_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@138)

Address: 11976

```
property ain138_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@138)

Address: 12276

```
property ain138_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN138_EF_INDEX (AIN#(0:149)_EF_INDEX@138)

Address: 9276

```
property ain138_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN138_EF_READ_A (AIN#(0:149)_EF_READ_A@138)

Address: 7276

```
property ain138_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN138_EF_READ_B (AIN#(0:149)_EF_READ_B@138)

Address: 7576

```
property ain138_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN138_EF_READ_C (AIN#(0:149)_EF_READ_C@138)

Address: 7876

```
property ain138_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN138_EF_READ_D (AIN#(0:149)_EF_READ_D@138)

Address: 8176

```
property ain139_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN139_BINARY (AIN#(0:254)_BINARY@139)

Address: 50278

```
property ain139_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@139)

Address: 9578

```
property ain139_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@139)

Address: 9878

```
property ain139_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@139)

Address: 10178

```
property ain139_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@139)

Address: 10478

```
property ain139_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@139)

Address: 10778

```
property ain139_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@139)

Address: 11078

```
property ain139_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@139)

Address: 11378

```
property ain139_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@139)

Address: 11678

```
property ain139_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@139)

Address: 11978

```
property ain139_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@139)

Address: 12278

```
property ain139_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN139_EF_INDEX (AIN#(0:149)_EF_INDEX@139)

Address: 9278

```
property ain139_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN139_EF_READ_A (AIN#(0:149)_EF_READ_A@139)

Address: 7278

```
property ain139_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN139_EF_READ_B (AIN#(0:149)_EF_READ_B@139)

Address: 7578

```
property ain139_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN139_EF_READ_C (AIN#(0:149)_EF_READ_C@139)

Address: 7878

```
property ain139_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN139_EF_READ_D (AIN#(0:149)_EF_READ_D@139)

Address: 8178

```
property ain13_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN13_BINARY (AIN#(0:254)_BINARY@13)

Address: 50026

```
property ain13_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@13)

Address: 9326

```
property ain13_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@13)

Address: 9626

```
property ain13_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@13)

Address: 9926

```
property ain13_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@13)

Address: 10226

```
property ain13_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@13)

Address: 10526

```
property ain13_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@13)

Address: 10826

```
property ain13_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@13)

Address: 11126

```
property ain13_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@13)

Address: 11426

```
property ain13_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@13)

Address: 11726

```
property ain13_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@13)

Address: 12026

```
property ain13_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN13_EF_INDEX (AIN#(0:149)_EF_INDEX@13)

Address: 9026

```
property ain13_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN13_EF_READ_A (AIN#(0:149)_EF_READ_A@13)

Address: 7026

```
property ain13_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN13_EF_READ_B (AIN#(0:149)_EF_READ_B@13)

Address: 7326

```
property ain13_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN13_EF_READ_C (AIN#(0:149)_EF_READ_C@13)

Address: 7626

```
property ain13_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN13_EF_READ_D (AIN#(0:149)_EF_READ_D@13)

Address: 7926

```
property ain140_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN140_BINARY (AIN#(0:254)_BINARY@140)

Address: 50280

```
property ain140_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@140)

Address: 9580

```
property ain140_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@140)

Address: 9880

```
property ain140_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@140)

Address: 10180

```
property ain140_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@140)

Address: 10480

```
property ain140_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@140)

Address: 10780

```
property ain140_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@140)

Address: 11080

```
property ain140_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@140)

Address: 11380

```
property ain140_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@140)

Address: 11680

```
property ain140_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@140)

Address: 11980

```
property ain140_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@140)

Address: 12280

```
property ain140_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN140_EF_INDEX (AIN#(0:149)_EF_INDEX@140)

Address: 9280

```
property ain140_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN140_EF_READ_A (AIN#(0:149)_EF_READ_A@140)

Address: 7280

```
property ain140_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN140_EF_READ_B (AIN#(0:149)_EF_READ_B@140)

Address: 7580

```
property ain140_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN140_EF_READ_C (AIN#(0:149)_EF_READ_C@140)

Address: 7880

```
property ain140_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN140_EF_READ_D (AIN#(0:149)_EF_READ_D@140)

Address: 8180

```
property ain141_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN141_BINARY (AIN#(0:254)_BINARY@141)

Address: 50282

```
property ain141_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@141)

Address: 9582

```
property ain141_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@141)

Address: 9882

```
property ain141_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@141)

Address: 10182

```
property ain141_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@141)

Address: 10482

```
property ain141_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@141)

Address: 10782

```
property ain141_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@141)

Address: 11082

```
property ain141_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@141)

Address: 11382

```
property ain141_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@141)

Address: 11682

```
property ain141_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@141)

Address: 11982

```
property ain141_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@141)

Address: 12282

```
property ain141_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN141_EF_INDEX (AIN#(0:149)_EF_INDEX@141)

Address: 9282

```
property ain141_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN141_EF_READ_A (AIN#(0:149)_EF_READ_A@141)

Address: 7282

```
property ain141_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN141_EF_READ_B (AIN#(0:149)_EF_READ_B@141)

Address: 7582

```
property ain141_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN141_EF_READ_C (AIN#(0:149)_EF_READ_C@141)

Address: 7882

```
property ain141_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN141_EF_READ_D (AIN#(0:149)_EF_READ_D@141)

Address: 8182

```
property ain142_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN142_BINARY (AIN#(0:254)_BINARY@142)

Address: 50284

```
property ain142_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@142)

Address: 9584

```
property ain142_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@142)

Address: 9884

```
property ain142_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@142)

Address: 10184

```
property ain142_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@142)

Address: 10484

```
property ain142_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@142)

Address: 10784

```
property ain142_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@142)

Address: 11084

```
property ain142_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@142)

Address: 11384

```
property ain142_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@142)

Address: 11684

```
property ain142_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@142)

Address: 11984

```
property ain142_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@142)

Address: 12284

```
property ain142_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN142_EF_INDEX (AIN#(0:149)_EF_INDEX@142)

Address: 9284

```
property ain142_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN142_EF_READ_A (AIN#(0:149)_EF_READ_A@142)

Address: 7284

```
property ain142_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN142_EF_READ_B (AIN#(0:149)_EF_READ_B@142)

Address: 7584

```
property ain142_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN142_EF_READ_C (AIN#(0:149)_EF_READ_C@142)

Address: 7884

```
property ain142_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN142_EF_READ_D (AIN#(0:149)_EF_READ_D@142)

Address: 8184

```
property ain143_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN143_BINARY (AIN#(0:254)_BINARY@143)

Address: 50286

```
property ain143_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@143)

Address: 9586

```
property ain143_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@143)

Address: 9886

```
property ain143_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@143)

Address: 10186

```
property ain143_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@143)

Address: 10486

```
property ain143_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@143)

Address: 10786

```
property ain143_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@143)

Address: 11086

```
property ain143_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@143)

Address: 11386

```
property ain143_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@143)

Address: 11686

```
property ain143_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@143)

Address: 11986

```
property ain143_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@143)

Address: 12286

```
property ain143_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN143_EF_INDEX (AIN#(0:149)_EF_INDEX@143)

Address: 9286

```
property ain143_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN143_EF_READ_A (AIN#(0:149)_EF_READ_A@143)

Address: 7286

```
property ain143_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN143_EF_READ_B (AIN#(0:149)_EF_READ_B@143)

Address: 7586

```
property ain143_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN143_EF_READ_C (AIN#(0:149)_EF_READ_C@143)

Address: 7886

```
property ain143_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN143_EF_READ_D (AIN#(0:149)_EF_READ_D@143)

Address: 8186

```
property ain144_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN144_BINARY (AIN#(0:254)_BINARY@144)

Address: 50288

```
property ain144_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@144)

Address: 9588

```
property ain144_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@144)

Address: 9888

```
property ain144_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@144)

Address: 10188

```
property ain144_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@144)

Address: 10488

```
property ain144_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@144)

Address: 10788

```
property ain144_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@144)

Address: 11088

```
property ain144_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@144)

Address: 11388

```
property ain144_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@144)

Address: 11688

```
property ain144_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@144)

Address: 11988

```
property ain144_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@144)

Address: 12288

```
property ain144_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN144_EF_INDEX (AIN#(0:149)_EF_INDEX@144)

Address: 9288

```
property ain144_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN144_EF_READ_A (AIN#(0:149)_EF_READ_A@144)

Address: 7288

```
property ain144_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN144_EF_READ_B (AIN#(0:149)_EF_READ_B@144)

Address: 7588

```
property ain144_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN144_EF_READ_C (AIN#(0:149)_EF_READ_C@144)

Address: 7888

```
property ain144_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN144_EF_READ_D (AIN#(0:149)_EF_READ_D@144)

Address: 8188

```
property ain145_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN145_BINARY (AIN#(0:254)_BINARY@145)

Address: 50290

```
property ain145_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@145)

Address: 9590

```
property ain145_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@145)

Address: 9890

```
property ain145_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@145)

Address: 10190

```
property ain145_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@145)

Address: 10490

```
property ain145_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@145)

Address: 10790

```
property ain145_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@145)

Address: 11090

```
property ain145_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@145)

Address: 11390

```
property ain145_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@145)

Address: 11690

```
property ain145_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@145)

Address: 11990

```
property ain145_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@145)

Address: 12290

```
property ain145_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN145_EF_INDEX (AIN#(0:149)_EF_INDEX@145)

Address: 9290

```
property ain145_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN145_EF_READ_A (AIN#(0:149)_EF_READ_A@145)

Address: 7290

```
property ain145_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN145_EF_READ_B (AIN#(0:149)_EF_READ_B@145)

Address: 7590

```
property ain145_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN145_EF_READ_C (AIN#(0:149)_EF_READ_C@145)

Address: 7890

```
property ain145_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN145_EF_READ_D (AIN#(0:149)_EF_READ_D@145)

Address: 8190

```
property ain146_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN146_BINARY (AIN#(0:254)_BINARY@146)

Address: 50292

```
property ain146_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@146)

Address: 9592

```
property ain146_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@146)

Address: 9892

```
property ain146_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@146)

Address: 10192

```
property ain146_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@146)

Address: 10492

```
property ain146_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@146)

Address: 10792

```
property ain146_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@146)

Address: 11092

```
property ain146_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@146)

Address: 11392

```
property ain146_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@146)

Address: 11692

```
property ain146_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@146)

Address: 11992

```
property ain146_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@146)

Address: 12292

```
property ain146_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN146_EF_INDEX (AIN#(0:149)_EF_INDEX@146)

Address: 9292

```
property ain146_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN146_EF_READ_A (AIN#(0:149)_EF_READ_A@146)

Address: 7292

```
property ain146_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN146_EF_READ_B (AIN#(0:149)_EF_READ_B@146)

Address: 7592

```
property ain146_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN146_EF_READ_C (AIN#(0:149)_EF_READ_C@146)

Address: 7892

```
property ain146_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN146_EF_READ_D (AIN#(0:149)_EF_READ_D@146)

Address: 8192

```
property ain147_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN147_BINARY (AIN#(0:254)_BINARY@147)

Address: 50294

```
property ain147_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@147)

Address: 9594

```
property ain147_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@147)

Address: 9894

```
property ain147_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@147)

Address: 10194

```
property ain147_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@147)

Address: 10494

```
property ain147_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@147)

Address: 10794

```
property ain147_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@147)

Address: 11094

```
property ain147_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@147)

Address: 11394

```
property ain147_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@147)

Address: 11694

```
property ain147_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@147)

Address: 11994

```
property ain147_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@147)

Address: 12294

```
property ain147_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN147_EF_INDEX (AIN#(0:149)_EF_INDEX@147)

Address: 9294

```
property ain147_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN147_EF_READ_A (AIN#(0:149)_EF_READ_A@147)

Address: 7294

```
property ain147_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN147_EF_READ_B (AIN#(0:149)_EF_READ_B@147)

Address: 7594

```
property ain147_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN147_EF_READ_C (AIN#(0:149)_EF_READ_C@147)

Address: 7894

```
property ain147_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN147_EF_READ_D (AIN#(0:149)_EF_READ_D@147)

Address: 8194

```
property ain148_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN148_BINARY (AIN#(0:254)_BINARY@148)

Address: 50296

```
property ain148_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@148)

Address: 9596

```
property ain148_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@148)

Address: 9896

```
property ain148_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@148)

Address: 10196

```
property ain148_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@148)

Address: 10496

```
property ain148_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@148)

Address: 10796

```
property ain148_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@148)

Address: 11096

```
property ain148_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@148)

Address: 11396

```
property ain148_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@148)

Address: 11696

```
property ain148_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@148)

Address: 11996

```
property ain148_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@148)

Address: 12296

```
property ain148_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN148_EF_INDEX (AIN#(0:149)_EF_INDEX@148)

Address: 9296

```
property ain148_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN148_EF_READ_A (AIN#(0:149)_EF_READ_A@148)

Address: 7296

```
property ain148_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN148_EF_READ_B (AIN#(0:149)_EF_READ_B@148)

Address: 7596

```
property ain148_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN148_EF_READ_C (AIN#(0:149)_EF_READ_C@148)

Address: 7896

```
property ain148_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN148_EF_READ_D (AIN#(0:149)_EF_READ_D@148)

Address: 8196

```
property ain149_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN149_BINARY (AIN#(0:254)_BINARY@149)

Address: 50298

```
property ain149_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@149)

Address: 9598

```
property ain149_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@149)

Address: 9898

```
property ain149_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@149)

Address: 10198

```
property ain149_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@149)

Address: 10498

```
property ain149_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@149)

Address: 10798

```
property ain149_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@149)

Address: 11098

```
property ain149_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@149)

Address: 11398

```
property ain149_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@149)

Address: 11698

```
property ain149_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@149)

Address: 11998

```
property ain149_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@149)

Address: 12298

```
property ain149_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN149_EF_INDEX (AIN#(0:149)_EF_INDEX@149)

Address: 9298

```
property ain149_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN149_EF_READ_A (AIN#(0:149)_EF_READ_A@149)

Address: 7298

```
property ain149_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN149_EF_READ_B (AIN#(0:149)_EF_READ_B@149)

Address: 7598

```
property ain149_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN149_EF_READ_C (AIN#(0:149)_EF_READ_C@149)

Address: 7898

```
property ain149_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN149_EF_READ_D (AIN#(0:149)_EF_READ_D@149)

Address: 8198

```
property ain14_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN14_BINARY (AIN#(0:254)_BINARY@14)

Address: 50028

```
property ain14_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@14)

Address: 9328

```
property ain14_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@14)

Address: 9628

```
property ain14_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@14)

Address: 9928

```
property ain14_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@14)

Address: 10228

```
property ain14_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@14)

Address: 10528

```
property ain14_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@14)

Address: 10828

```
property ain14_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@14)

Address: 11128

```
property ain14_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@14)

Address: 11428

```
property ain14_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@14)

Address: 11728

```
property ain14_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@14)

Address: 12028

```
property ain14_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN14_EF_INDEX (AIN#(0:149)_EF_INDEX@14)

Address: 9028

```
property ain14_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN14_EF_READ_A (AIN#(0:149)_EF_READ_A@14)

Address: 7028

```
property ain14_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN14_EF_READ_B (AIN#(0:149)_EF_READ_B@14)

Address: 7328

```
property ain14_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN14_EF_READ_C (AIN#(0:149)_EF_READ_C@14)

Address: 7628

```
property ain14_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN14_EF_READ_D (AIN#(0:149)_EF_READ_D@14)

Address: 7928

```
property ain150_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN150_BINARY (AIN#(0:254)_BINARY@150)

Address: 50300

```
property ain151_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN151_BINARY (AIN#(0:254)_BINARY@151)

Address: 50302

```
property ain152_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN152_BINARY (AIN#(0:254)_BINARY@152)

Address: 50304

```
property ain153_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN153_BINARY (AIN#(0:254)_BINARY@153)

Address: 50306

```
property ain154_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN154_BINARY (AIN#(0:254)_BINARY@154)

Address: 50308

```
property ain155_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN155_BINARY (AIN#(0:254)_BINARY@155)

Address: 50310

```
property ain156_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN156_BINARY (AIN#(0:254)_BINARY@156)

Address: 50312

```
property ain157_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN157_BINARY (AIN#(0:254)_BINARY@157)

Address: 50314

```
property ain158_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN158_BINARY (AIN#(0:254)_BINARY@158)

Address: 50316

```
property ain159_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN159_BINARY (AIN#(0:254)_BINARY@159)

Address: 50318

```
property ain15_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN15_BINARY (AIN#(0:254)_BINARY@15)

Address: 50030

```
property ain15_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@15)

Address: 9330

```
property ain15_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@15)

Address: 9630

```
property ain15_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@15)

Address: 9930

```
property ain15_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@15)

Address: 10230

```
property ain15_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@15)

Address: 10530

```
property ain15_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@15)

Address: 10830

```
property ain15_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@15)

Address: 11130

```
property ain15_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@15)

Address: 11430

```
property ain15_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@15)

Address: 11730

```
property ain15_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@15)

Address: 12030

```
property ain15_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN15_EF_INDEX (AIN#(0:149)_EF_INDEX@15)

Address: 9030

```
property ain15_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN15_EF_READ_A (AIN#(0:149)_EF_READ_A@15)

Address: 7030

```
property ain15_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN15_EF_READ_B (AIN#(0:149)_EF_READ_B@15)

Address: 7330

```
property ain15_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN15_EF_READ_C (AIN#(0:149)_EF_READ_C@15)

Address: 7630

```
property ain15_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN15_EF_READ_D (AIN#(0:149)_EF_READ_D@15)

Address: 7930

```
property ain160_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN160_BINARY (AIN#(0:254)_BINARY@160)

Address: 50320

```
property ain161_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN161_BINARY (AIN#(0:254)_BINARY@161)

Address: 50322

```
property ain162_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN162_BINARY (AIN#(0:254)_BINARY@162)

Address: 50324

```
property ain163_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN163_BINARY (AIN#(0:254)_BINARY@163)

Address: 50326

```
property ain164_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN164_BINARY (AIN#(0:254)_BINARY@164)

Address: 50328

```
property ain165_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN165_BINARY (AIN#(0:254)_BINARY@165)

Address: 50330

```
property ain166_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN166_BINARY (AIN#(0:254)_BINARY@166)

Address: 50332

```
property ain167_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN167_BINARY (AIN#(0:254)_BINARY@167)

Address: 50334

```
property ain168_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN168_BINARY (AIN#(0:254)_BINARY@168)

Address: 50336

```
property ain169_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN169_BINARY (AIN#(0:254)_BINARY@169)

Address: 50338

```
property ain16_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN16_BINARY (AIN#(0:254)_BINARY@16)

Address: 50032

```
property ain16_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@16)

Address: 9332

```
property ain16_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@16)

Address: 9632

```
property ain16_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@16)

Address: 9932

```
property ain16_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@16)

Address: 10232

```
property ain16_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@16)

Address: 10532

```
property ain16_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@16)

Address: 10832

```
property ain16_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@16)

Address: 11132

```
property ain16_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@16)

Address: 11432

```
property ain16_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@16)

Address: 11732

```
property ain16_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@16)

Address: 12032

```
property ain16_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN16_EF_INDEX (AIN#(0:149)_EF_INDEX@16)

Address: 9032

```
property ain16_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN16_EF_READ_A (AIN#(0:149)_EF_READ_A@16)

Address: 7032

```
property ain16_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN16_EF_READ_B (AIN#(0:149)_EF_READ_B@16)

Address: 7332

```
property ain16_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN16_EF_READ_C (AIN#(0:149)_EF_READ_C@16)

Address: 7632

```
property ain16_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN16_EF_READ_D (AIN#(0:149)_EF_READ_D@16)

Address: 7932

```
property ain170_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN170_BINARY (AIN#(0:254)_BINARY@170)

Address: 50340

```
property ain171_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN171_BINARY (AIN#(0:254)_BINARY@171)

Address: 50342

```
property ain172_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN172_BINARY (AIN#(0:254)_BINARY@172)

Address: 50344

```
property ain173_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN173_BINARY (AIN#(0:254)_BINARY@173)

Address: 50346

```
property ain174_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN174_BINARY (AIN#(0:254)_BINARY@174)

Address: 50348

```
property ain175_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN175_BINARY (AIN#(0:254)_BINARY@175)

Address: 50350

```
property ain176_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN176_BINARY (AIN#(0:254)_BINARY@176)

Address: 50352

```
property ain177_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN177_BINARY (AIN#(0:254)_BINARY@177)

Address: 50354

```
property ain178_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN178_BINARY (AIN#(0:254)_BINARY@178)

Address: 50356

```
property ain179_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN179_BINARY (AIN#(0:254)_BINARY@179)

Address: 50358

```
property ain17_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN17_BINARY (AIN#(0:254)_BINARY@17)

Address: 50034

```
property ain17_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@17)

Address: 9334

```
property ain17_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@17)

Address: 9634

```
property ain17_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@17)

Address: 9934

```
property ain17_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@17)

Address: 10234

```
property ain17_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@17)

Address: 10534

```
property ain17_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@17)

Address: 10834

```
property ain17_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@17)

Address: 11134

```
property ain17_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@17)

Address: 11434

```
property ain17_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@17)

Address: 11734

```
property ain17_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@17)

Address: 12034

```
property ain17_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN17_EF_INDEX (AIN#(0:149)_EF_INDEX@17)

Address: 9034

```
property ain17_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN17_EF_READ_A (AIN#(0:149)_EF_READ_A@17)

Address: 7034

```
property ain17_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN17_EF_READ_B (AIN#(0:149)_EF_READ_B@17)

Address: 7334

```
property ain17_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN17_EF_READ_C (AIN#(0:149)_EF_READ_C@17)

Address: 7634

```
property ain17_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN17_EF_READ_D (AIN#(0:149)_EF_READ_D@17)

Address: 7934

```
property ain180_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN180_BINARY (AIN#(0:254)_BINARY@180)

Address: 50360

```
property ain181_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN181_BINARY (AIN#(0:254)_BINARY@181)

Address: 50362

```
property ain182_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN182_BINARY (AIN#(0:254)_BINARY@182)

Address: 50364

```
property ain183_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN183_BINARY (AIN#(0:254)_BINARY@183)

Address: 50366

```
property ain184_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN184_BINARY (AIN#(0:254)_BINARY@184)

Address: 50368

```
property ain185_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN185_BINARY (AIN#(0:254)_BINARY@185)

Address: 50370

```
property ain186_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN186_BINARY (AIN#(0:254)_BINARY@186)

Address: 50372

```
property ain187_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN187_BINARY (AIN#(0:254)_BINARY@187)

Address: 50374

```
property ain188_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN188_BINARY (AIN#(0:254)_BINARY@188)

Address: 50376

```
property ain189_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN189_BINARY (AIN#(0:254)_BINARY@189)

Address: 50378

```
property ain18_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN18_BINARY (AIN#(0:254)_BINARY@18)

Address: 50036

```
property ain18_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@18)

Address: 9336

```
property ain18_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@18)

Address: 9636

```
property ain18_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@18)

Address: 9936

```
property ain18_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@18)

Address: 10236

```
property ain18_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@18)

Address: 10536

```
property ain18_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@18)

Address: 10836

```
property ain18_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@18)

Address: 11136

```
property ain18_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@18)

Address: 11436

```
property ain18_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@18)

Address: 11736

```
property ain18_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@18)

Address: 12036

```
property ain18_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN18_EF_INDEX (AIN#(0:149)_EF_INDEX@18)

Address: 9036

```
property ain18_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN18_EF_READ_A (AIN#(0:149)_EF_READ_A@18)

Address: 7036

```
property ain18_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN18_EF_READ_B (AIN#(0:149)_EF_READ_B@18)

Address: 7336

```
property ain18_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN18_EF_READ_C (AIN#(0:149)_EF_READ_C@18)

Address: 7636

```
property ain18_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN18_EF_READ_D (AIN#(0:149)_EF_READ_D@18)

Address: 7936

```
property ain190_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN190_BINARY (AIN#(0:254)_BINARY@190)

Address: 50380

```
property ain191_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN191_BINARY (AIN#(0:254)_BINARY@191)

Address: 50382

```
property ain192_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN192_BINARY (AIN#(0:254)_BINARY@192)

Address: 50384

```
property ain193_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN193_BINARY (AIN#(0:254)_BINARY@193)

Address: 50386

```
property ain194_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN194_BINARY (AIN#(0:254)_BINARY@194)

Address: 50388

```
property ain195_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN195_BINARY (AIN#(0:254)_BINARY@195)

Address: 50390

```
property ain196_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN196_BINARY (AIN#(0:254)_BINARY@196)

Address: 50392

```
property ain197_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN197_BINARY (AIN#(0:254)_BINARY@197)

Address: 50394

```
property ain198_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN198_BINARY (AIN#(0:254)_BINARY@198)

Address: 50396

```
property ain199_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN199_BINARY (AIN#(0:254)_BINARY@199)

Address: 50398

```
property ain19_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN19_BINARY (AIN#(0:254)_BINARY@19)

Address: 50038

```
property ain19_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@19)

Address: 9338

```
property ain19_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@19)

Address: 9638

```
property ain19_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@19)

Address: 9938

```
property ain19_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@19)

Address: 10238

```
property ain19_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@19)

Address: 10538

```
property ain19_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@19)

Address: 10838

```
property ain19_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@19)

Address: 11138

```
property ain19_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@19)

Address: 11438

```
property ain19_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@19)

Address: 11738

```
property ain19_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@19)

Address: 12038

```
property ain19_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN19_EF_INDEX (AIN#(0:149)_EF_INDEX@19)

Address: 9038

```
property ain19_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN19_EF_READ_A (AIN#(0:149)_EF_READ_A@19)

Address: 7038

```
property ain19_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN19_EF_READ_B (AIN#(0:149)_EF_READ_B@19)

Address: 7338

```
property ain19_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN19_EF_READ_C (AIN#(0:149)_EF_READ_C@19)

Address: 7638

```
property ain19_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN19_EF_READ_D (AIN#(0:149)_EF_READ_D@19)

Address: 7938

```
property ain1_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN1_BINARY (AIN#(0:254)_BINARY@1)

Address: 50002

```
property ain1_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@1)

Address: 9302

```
property ain1_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@1)

Address: 9602

```
property ain1_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@1)

Address: 9902

```
property ain1_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@1)

Address: 10202

```
property ain1_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@1)

Address: 10502

```
property ain1_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@1)

Address: 10802

```
property ain1_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@1)

Address: 11102

```
property ain1_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@1)

Address: 11402

```
property ain1_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@1)

Address: 11702

```
property ain1_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@1)

Address: 12002

```
property ain1_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN1_EF_INDEX (AIN#(0:149)_EF_INDEX@1)

Address: 9002

```
property ain1_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN1_EF_READ_A (AIN#(0:149)_EF_READ_A@1)

Address: 7002

```
property ain1_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN1_EF_READ_B (AIN#(0:149)_EF_READ_B@1)

Address: 7302

```
property ain1_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN1_EF_READ_C (AIN#(0:149)_EF_READ_C@1)

Address: 7602

```
property ain1_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN1_EF_READ_D (AIN#(0:149)_EF_READ_D@1)

Address: 7902

```
property ain1_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN1_NEGATIVE_CH (AIN#(0:3)_NEGATIVE_CH@1)

Address: 41001

```
property ain1_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN1_RANGE (AIN#(0:3)_RANGE@1)

Address: 40002

```
property ain1_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN1_RESOLUTION_INDEX (AIN#(0:3)_RESOLUTION_INDEX@1)

Address: 41501

```
property ain1_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN1_SETTLING_US (AIN#(0:3)_SETTLING_US@1)

Address: 42002

```
property ain2: float
```

Returns the voltage of the specified analog input.

Register: AIN2 (AIN#(0:11)@2)

Address: 4

```
property ain200_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN200_BINARY (AIN#(0:254)_BINARY@200)

Address: 50400

```
property ain201_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN201_BINARY (AIN#(0:254)_BINARY@201)

Address: 50402

```
property ain202_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN202_BINARY (AIN#(0:254)_BINARY@202)

Address: 50404

```
property ain203_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN203_BINARY (AIN#(0:254)_BINARY@203)

Address: 50406

```
property ain204_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN204_BINARY (AIN#(0:254)_BINARY@204)

Address: 50408

```
property ain205_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN205_BINARY (AIN#(0:254)_BINARY@205)

Address: 50410

```
property ain206_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN206_BINARY (AIN#(0:254)_BINARY@206)

Address: 50412

```
property ain207_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN207_BINARY (AIN#(0:254)_BINARY@207)

Address: 50414

```
property ain208_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN208_BINARY (AIN#(0:254)_BINARY@208)

Address: 50416

```
property ain209_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN209_BINARY (AIN#(0:254)_BINARY@209)

Address: 50418

```
property ain20_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN20_BINARY (AIN#(0:254)_BINARY@20)

Address: 50040

```
property ain20_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@20)

Address: 9340

```
property ain20_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@20)

Address: 9640

```
property ain20_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@20)

Address: 9940

```
property ain20_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@20)

Address: 10240

```
property ain20_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@20)

Address: 10540

```
property ain20_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@20)

Address: 10840

```
property ain20_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@20)

Address: 11140

```
property ain20_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@20)

Address: 11440

```
property ain20_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@20)

Address: 11740

```
property ain20_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@20)

Address: 12040

```
property ain20_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN20_EF_INDEX (AIN#(0:149)_EF_INDEX@20)

Address: 9040

```
property ain20_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN20_EF_READ_A (AIN#(0:149)_EF_READ_A@20)

Address: 7040

```
property ain20_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN20_EF_READ_B (AIN#(0:149)_EF_READ_B@20)

Address: 7340

```
property ain20_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN20_EF_READ_C (AIN#(0:149)_EF_READ_C@20)

Address: 7640

```
property ain20_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN20_EF_READ_D (AIN#(0:149)_EF_READ_D@20)

Address: 7940

```
property ain210_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN210_BINARY (AIN#(0:254)_BINARY@210)

Address: 50420

```
property ain211_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN211_BINARY (AIN#(0:254)_BINARY@211)

Address: 50422

```
property ain212_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN212_BINARY (AIN#(0:254)_BINARY@212)

Address: 50424

```
property ain213_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN213_BINARY (AIN#(0:254)_BINARY@213)

Address: 50426

```
property ain214_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN214_BINARY (AIN#(0:254)_BINARY@214)

Address: 50428

```
property ain215_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN215_BINARY (AIN#(0:254)_BINARY@215)

Address: 50430

```
property ain216_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN216_BINARY (AIN#(0:254)_BINARY@216)

Address: 50432

```
property ain217_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN217_BINARY (AIN#(0:254)_BINARY@217)

Address: 50434

```
property ain218_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN218_BINARY (AIN#(0:254)_BINARY@218)

Address: 50436

```
property ain219_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN219_BINARY (AIN#(0:254)_BINARY@219)

Address: 50438

```
property ain21_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN21_BINARY (AIN#(0:254)_BINARY@21)

Address: 50042

```
property ain21_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@21)

Address: 9342

```
property ain21_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@21)

Address: 9642

```
property ain21_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@21)

Address: 9942

```
property ain21_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@21)

Address: 10242

```
property ain21_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@21)

Address: 10542

```
property ain21_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@21)

Address: 10842

```
property ain21_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@21)

Address: 11142

```
property ain21_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@21)

Address: 11442

```
property ain21_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@21)

Address: 11742

```
property ain21_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@21)

Address: 12042

```
property ain21_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN21_EF_INDEX (AIN#(0:149)_EF_INDEX@21)

Address: 9042

```
property ain21_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN21_EF_READ_A (AIN#(0:149)_EF_READ_A@21)

Address: 7042

```
property ain21_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN21_EF_READ_B (AIN#(0:149)_EF_READ_B@21)

Address: 7342

```
property ain21_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN21_EF_READ_C (AIN#(0:149)_EF_READ_C@21)

Address: 7642

```
property ain21_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN21_EF_READ_D (AIN#(0:149)_EF_READ_D@21)

Address: 7942

```
property ain220_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN220_BINARY (AIN#(0:254)_BINARY@220)

Address: 50440

```
property ain221_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN221_BINARY (AIN#(0:254)_BINARY@221)

Address: 50442

```
property ain222_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN222_BINARY (AIN#(0:254)_BINARY@222)

Address: 50444

```
property ain223_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN223_BINARY (AIN#(0:254)_BINARY@223)

Address: 50446

```
property ain224_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN224_BINARY (AIN#(0:254)_BINARY@224)

Address: 50448

```
property ain225_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN225_BINARY (AIN#(0:254)_BINARY@225)

Address: 50450

```
property ain226_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN226_BINARY (AIN#(0:254)_BINARY@226)

Address: 50452

```
property ain227_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN227_BINARY (AIN#(0:254)_BINARY@227)

Address: 50454

```
property ain228_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN228_BINARY (AIN#(0:254)_BINARY@228)

Address: 50456

```
property ain229_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN229_BINARY (AIN#(0:254)_BINARY@229)

Address: 50458

```
property ain22_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN22_BINARY (AIN#(0:254)_BINARY@22)

Address: 50044

```
property ain22_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@22)

Address: 9344

```
property ain22_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@22)

Address: 9644

```
property ain22_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@22)

Address: 9944

```
property ain22_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@22)

Address: 10244

```
property ain22_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@22)

Address: 10544

```
property ain22_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@22)

Address: 10844

```
property ain22_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@22)

Address: 11144

```
property ain22_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@22)

Address: 11444

```
property ain22_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@22)

Address: 11744

```
property ain22_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@22)

Address: 12044

```
property ain22_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN22_EF_INDEX (AIN#(0:149)_EF_INDEX@22)

Address: 9044

```
property ain22_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN22_EF_READ_A (AIN#(0:149)_EF_READ_A@22)

Address: 7044

```
property ain22_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN22_EF_READ_B (AIN#(0:149)_EF_READ_B@22)

Address: 7344

```
property ain22_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN22_EF_READ_C (AIN#(0:149)_EF_READ_C@22)

Address: 7644

```
property ain22_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN22_EF_READ_D (AIN#(0:149)_EF_READ_D@22)

Address: 7944

```
property ain230_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN230_BINARY (AIN#(0:254)_BINARY@230)

Address: 50460

```
property ain231_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN231_BINARY (AIN#(0:254)_BINARY@231)

Address: 50462

```
property ain232_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN232_BINARY (AIN#(0:254)_BINARY@232)

Address: 50464

```
property ain233_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN233_BINARY (AIN#(0:254)_BINARY@233)

Address: 50466

```
property ain234_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN234_BINARY (AIN#(0:254)_BINARY@234)

Address: 50468

```
property ain235_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN235_BINARY (AIN#(0:254)_BINARY@235)

Address: 50470

```
property ain236_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN236_BINARY (AIN#(0:254)_BINARY@236)

Address: 50472

```
property ain237_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN237_BINARY (AIN#(0:254)_BINARY@237)

Address: 50474

```
property ain238_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN238_BINARY (AIN#(0:254)_BINARY@238)

Address: 50476

```
property ain239_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN239_BINARY (AIN#(0:254)_BINARY@239)

Address: 50478

```
property ain23_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN23_BINARY (AIN#(0:254)_BINARY@23)

Address: 50046

```
property ain23_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@23)

Address: 9346

```
property ain23_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@23)

Address: 9646

```
property ain23_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@23)

Address: 9946

```
property ain23_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@23)

Address: 10246

```
property ain23_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@23)

Address: 10546

```
property ain23_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@23)

Address: 10846

```
property ain23_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@23)

Address: 11146

```
property ain23_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@23)

Address: 11446

```
property ain23_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@23)

Address: 11746

```
property ain23_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@23)

Address: 12046

```
property ain23_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN23_EF_INDEX (AIN#(0:149)_EF_INDEX@23)

Address: 9046

```
property ain23_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN23_EF_READ_A (AIN#(0:149)_EF_READ_A@23)

Address: 7046

```
property ain23_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN23_EF_READ_B (AIN#(0:149)_EF_READ_B@23)

Address: 7346

```
property ain23_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN23_EF_READ_C (AIN#(0:149)_EF_READ_C@23)

Address: 7646

```
property ain23_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN23_EF_READ_D (AIN#(0:149)_EF_READ_D@23)

Address: 7946

```
property ain240_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN240_BINARY (AIN#(0:254)_BINARY@240)

Address: 50480

```
property ain241_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN241_BINARY (AIN#(0:254)_BINARY@241)

Address: 50482

```
property ain242_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN242_BINARY (AIN#(0:254)_BINARY@242)

Address: 50484

```
property ain243_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN243_BINARY (AIN#(0:254)_BINARY@243)

Address: 50486

```
property ain244_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN244_BINARY (AIN#(0:254)_BINARY@244)

Address: 50488

```
property ain245_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN245_BINARY (AIN#(0:254)_BINARY@245)

Address: 50490

```
property ain246_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN246_BINARY (AIN#(0:254)_BINARY@246)

Address: 50492

```
property ain247_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN247_BINARY (AIN#(0:254)_BINARY@247)

Address: 50494

```
property ain248_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN248_BINARY (AIN#(0:254)_BINARY@248)

Address: 50496

```
property ain249_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN249_BINARY (AIN#(0:254)_BINARY@249)

Address: 50498

```
property ain24_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN24_BINARY (AIN#(0:254)_BINARY@24)

Address: 50048

```
property ain24_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@24)

Address: 9348

```
property ain24_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@24)

Address: 9648

```
property ain24_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@24)

Address: 9948

```
property ain24_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@24)

Address: 10248

```
property ain24_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@24)

Address: 10548

```
property ain24_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@24)

Address: 10848

```
property ain24_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@24)

Address: 11148

```
property ain24_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@24)

Address: 11448

```
property ain24_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@24)

Address: 11748

```
property ain24_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@24)

Address: 12048

```
property ain24_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN24_EF_INDEX (AIN#(0:149)_EF_INDEX@24)

Address: 9048

```
property ain24_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN24_EF_READ_A (AIN#(0:149)_EF_READ_A@24)

Address: 7048

```
property ain24_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN24_EF_READ_B (AIN#(0:149)_EF_READ_B@24)

Address: 7348

```
property ain24_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN24_EF_READ_C (AIN#(0:149)_EF_READ_C@24)

Address: 7648

```
property ain24_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN24_EF_READ_D (AIN#(0:149)_EF_READ_D@24)

Address: 7948

```
property ain250_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN250_BINARY (AIN#(0:254)_BINARY@250)

Address: 50500

```
property ain251_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN251_BINARY (AIN#(0:254)_BINARY@251)

Address: 50502

```
property ain252_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN252_BINARY (AIN#(0:254)_BINARY@252)

Address: 50504

```
property ain253_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN253_BINARY (AIN#(0:254)_BINARY@253)

Address: 50506

```
property ain254_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN254_BINARY (AIN#(0:254)_BINARY@254)

Address: 50508

```
property ain25_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN25_BINARY (AIN#(0:254)_BINARY@25)

Address: 50050

```
property ain25_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@25)

Address: 9350

```
property ain25_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@25)

Address: 9650

```
property ain25_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@25)

Address: 9950

```
property ain25_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@25)

Address: 10250

```
property ain25_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@25)

Address: 10550

```
property ain25_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@25)

Address: 10850

```
property ain25_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@25)

Address: 11150

```
property ain25_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@25)

Address: 11450

```
property ain25_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@25)

Address: 11750

```
property ain25_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@25)

Address: 12050

```
property ain25_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN25_EF_INDEX (AIN#(0:149)_EF_INDEX@25)

Address: 9050

```
property ain25_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN25_EF_READ_A (AIN#(0:149)_EF_READ_A@25)

Address: 7050

```
property ain25_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN25_EF_READ_B (AIN#(0:149)_EF_READ_B@25)

Address: 7350

```
property ain25_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN25_EF_READ_C (AIN#(0:149)_EF_READ_C@25)

Address: 7650

```
property ain25_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN25_EF_READ_D (AIN#(0:149)_EF_READ_D@25)

Address: 7950

```
property ain26_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN26_BINARY (AIN#(0:254)_BINARY@26)

Address: 50052

```
property ain26_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@26)

Address: 9352

```
property ain26_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@26)

Address: 9652

```
property ain26_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@26)

Address: 9952

```
property ain26_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@26)

Address: 10252

```
property ain26_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@26)

Address: 10552

```
property ain26_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@26)

Address: 10852

```
property ain26_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@26)

Address: 11152

```
property ain26_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@26)

Address: 11452

```
property ain26_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@26)

Address: 11752

```
property ain26_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@26)

Address: 12052

```
property ain26_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN26_EF_INDEX (AIN#(0:149)_EF_INDEX@26)

Address: 9052

```
property ain26_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN26_EF_READ_A (AIN#(0:149)_EF_READ_A@26)

Address: 7052

```
property ain26_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN26_EF_READ_B (AIN#(0:149)_EF_READ_B@26)

Address: 7352

```
property ain26_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN26_EF_READ_C (AIN#(0:149)_EF_READ_C@26)

Address: 7652

```
property ain26_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN26_EF_READ_D (AIN#(0:149)_EF_READ_D@26)

Address: 7952

```
property ain27_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN27_BINARY (AIN#(0:254)_BINARY@27)

Address: 50054

```
property ain27_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@27)

Address: 9354

```
property ain27_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@27)

Address: 9654

```
property ain27_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@27)

Address: 9954

```
property ain27_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@27)

Address: 10254

```
property ain27_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@27)

Address: 10554

```
property ain27_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@27)

Address: 10854

```
property ain27_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@27)

Address: 11154

```
property ain27_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@27)

Address: 11454

```
property ain27_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@27)

Address: 11754

```
property ain27_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@27)

Address: 12054

```
property ain27_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN27_EF_INDEX (AIN#(0:149)_EF_INDEX@27)

Address: 9054

```
property ain27_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN27_EF_READ_A (AIN#(0:149)_EF_READ_A@27)

Address: 7054

```
property ain27_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN27_EF_READ_B (AIN#(0:149)_EF_READ_B@27)

Address: 7354

```
property ain27_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN27_EF_READ_C (AIN#(0:149)_EF_READ_C@27)

Address: 7654

```
property ain27_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN27_EF_READ_D (AIN#(0:149)_EF_READ_D@27)

Address: 7954

```
property ain28_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN28_BINARY (AIN#(0:254)_BINARY@28)

Address: 50056

```
property ain28_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@28)

Address: 9356

```
property ain28_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@28)

Address: 9656

```
property ain28_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@28)

Address: 9956

```
property ain28_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@28)

Address: 10256

```
property ain28_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@28)

Address: 10556

```
property ain28_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@28)

Address: 10856

```
property ain28_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@28)

Address: 11156

```
property ain28_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@28)

Address: 11456

```
property ain28_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@28)

Address: 11756

```
property ain28_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@28)

Address: 12056

```
property ain28_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN28_EF_INDEX (AIN#(0:149)_EF_INDEX@28)

Address: 9056

```
property ain28_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN28_EF_READ_A (AIN#(0:149)_EF_READ_A@28)

Address: 7056

```
property ain28_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN28_EF_READ_B (AIN#(0:149)_EF_READ_B@28)

Address: 7356

```
property ain28_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN28_EF_READ_C (AIN#(0:149)_EF_READ_C@28)

Address: 7656

```
property ain28_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN28_EF_READ_D (AIN#(0:149)_EF_READ_D@28)

Address: 7956

```
property ain29_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN29_BINARY (AIN#(0:254)_BINARY@29)

Address: 50058

```
property ain29_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@29)

Address: 9358

```
property ain29_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@29)

Address: 9658

```
property ain29_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@29)

Address: 9958

```
property ain29_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@29)

Address: 10258

```
property ain29_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@29)

Address: 10558

```
property ain29_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@29)

Address: 10858

```
property ain29_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@29)

Address: 11158

```
property ain29_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@29)

Address: 11458

```
property ain29_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@29)

Address: 11758

```
property ain29_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@29)

Address: 12058

```
property ain29_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN29_EF_INDEX (AIN#(0:149)_EF_INDEX@29)

Address: 9058

```
property ain29_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN29_EF_READ_A (AIN#(0:149)_EF_READ_A@29)

Address: 7058

```
property ain29_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN29_EF_READ_B (AIN#(0:149)_EF_READ_B@29)

Address: 7358

```
property ain29_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN29_EF_READ_C (AIN#(0:149)_EF_READ_C@29)

Address: 7658

```
property ain29_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN29_EF_READ_D (AIN#(0:149)_EF_READ_D@29)

Address: 7958

```
property ain2_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN2_BINARY (AIN#(0:254)_BINARY@2)

Address: 50004

```
property ain2_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@2)

Address: 9304

```
property ain2_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@2)

Address: 9604

```
property ain2_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@2)

Address: 9904

```
property ain2_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@2)

Address: 10204

```
property ain2_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@2)

Address: 10504

```
property ain2_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@2)

Address: 10804

```
property ain2_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@2)

Address: 11104

```
property ain2_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@2)

Address: 11404

```
property ain2_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@2)

Address: 11704

```
property ain2_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@2)

Address: 12004

```
property ain2_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN2_EF_INDEX (AIN#(0:149)_EF_INDEX@2)

Address: 9004

```
property ain2_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN2_EF_READ_A (AIN#(0:149)_EF_READ_A@2)

Address: 7004

```
property ain2_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN2_EF_READ_B (AIN#(0:149)_EF_READ_B@2)

Address: 7304

```
property ain2_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN2_EF_READ_C (AIN#(0:149)_EF_READ_C@2)

Address: 7604

```
property ain2_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN2_EF_READ_D (AIN#(0:149)_EF_READ_D@2)

Address: 7904

```
property ain2_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN2_NEGATIVE_CH (AIN#(0:3)_NEGATIVE_CH@2)

Address: 41002

```
property ain2_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN2_RANGE (AIN#(0:3)_RANGE@2)

Address: 40004

```
property ain2_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN2_RESOLUTION_INDEX (AIN#(0:3)_RESOLUTION_INDEX@2)

Address: 41502

```
property ain2_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN2_SETTLING_US (AIN#(0:3)_SETTLING_US@2)

Address: 42004

```
property ain3: float
```

Returns the voltage of the specified analog input.

Register: AIN3 (AIN#(0:11)@3)

Address: 6

```
property ain30_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN30_BINARY (AIN#(0:254)_BINARY@30)

Address: 50060

```
property ain30_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@30)

Address: 9360

```
property ain30_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@30)

Address: 9660

```
property ain30_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@30)

Address: 9960

```
property ain30_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@30)

Address: 10260

```
property ain30_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@30)

Address: 10560

```
property ain30_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@30)

Address: 10860

```
property ain30_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@30)

Address: 11160

```
property ain30_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@30)

Address: 11460

```
property ain30_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@30)

Address: 11760

```
property ain30_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@30)

Address: 12060

```
property ain30_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN30_EF_INDEX (AIN#(0:149)_EF_INDEX@30)

Address: 9060

```
property ain30_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN30_EF_READ_A (AIN#(0:149)_EF_READ_A@30)

Address: 7060

```
property ain30_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN30_EF_READ_B (AIN#(0:149)_EF_READ_B@30)

Address: 7360

```
property ain30_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN30_EF_READ_C (AIN#(0:149)_EF_READ_C@30)

Address: 7660

```
property ain30_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN30_EF_READ_D (AIN#(0:149)_EF_READ_D@30)

Address: 7960

```
property ain31_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN31_BINARY (AIN#(0:254)_BINARY@31)

Address: 50062

```
property ain31_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@31)

Address: 9362

```
property ain31_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@31)

Address: 9662

```
property ain31_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@31)

Address: 9962

```
property ain31_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@31)

Address: 10262

```
property ain31_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@31)

Address: 10562

```
property ain31_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@31)

Address: 10862

```
property ain31_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@31)

Address: 11162

```
property ain31_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@31)

Address: 11462

```
property ain31_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@31)

Address: 11762

```
property ain31_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@31)

Address: 12062

```
property ain31_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN31_EF_INDEX (AIN#(0:149)_EF_INDEX@31)

Address: 9062

```
property ain31_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN31_EF_READ_A (AIN#(0:149)_EF_READ_A@31)

Address: 7062

```
property ain31_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN31_EF_READ_B (AIN#(0:149)_EF_READ_B@31)

Address: 7362

```
property ain31_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN31_EF_READ_C (AIN#(0:149)_EF_READ_C@31)

Address: 7662

```
property ain31_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN31_EF_READ_D (AIN#(0:149)_EF_READ_D@31)

Address: 7962

```
property ain32_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN32_BINARY (AIN#(0:254)_BINARY@32)

Address: 50064

```
property ain32_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@32)

Address: 9364

```
property ain32_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@32)

Address: 9664

```
property ain32_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@32)

Address: 9964

```
property ain32_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@32)

Address: 10264

```
property ain32_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@32)

Address: 10564

```
property ain32_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@32)

Address: 10864

```
property ain32_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@32)

Address: 11164

```
property ain32_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@32)

Address: 11464

```
property ain32_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@32)

Address: 11764

```
property ain32_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@32)

Address: 12064

```
property ain32_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN32_EF_INDEX (AIN#(0:149)_EF_INDEX@32)

Address: 9064

```
property ain32_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN32_EF_READ_A (AIN#(0:149)_EF_READ_A@32)

Address: 7064

```
property ain32_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN32_EF_READ_B (AIN#(0:149)_EF_READ_B@32)

Address: 7364

```
property ain32_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN32_EF_READ_C (AIN#(0:149)_EF_READ_C@32)

Address: 7664

```
property ain32_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN32_EF_READ_D (AIN#(0:149)_EF_READ_D@32)

Address: 7964

```
property ain33_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN33_BINARY (AIN#(0:254)_BINARY@33)

Address: 50066

```
property ain33_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@33)

Address: 9366

```
property ain33_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@33)

Address: 9666

```
property ain33_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@33)

Address: 9966

```
property ain33_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@33)

Address: 10266

```
property ain33_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@33)

Address: 10566

```
property ain33_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@33)

Address: 10866

```
property ain33_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@33)

Address: 11166

```
property ain33_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@33)

Address: 11466

```
property ain33_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@33)

Address: 11766

```
property ain33_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@33)

Address: 12066

```
property ain33_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN33_EF_INDEX (AIN#(0:149)_EF_INDEX@33)

Address: 9066

```
property ain33_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN33_EF_READ_A (AIN#(0:149)_EF_READ_A@33)

Address: 7066

```
property ain33_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN33_EF_READ_B (AIN#(0:149)_EF_READ_B@33)

Address: 7366

```
property ain33_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN33_EF_READ_C (AIN#(0:149)_EF_READ_C@33)

Address: 7666

```
property ain33_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN33_EF_READ_D (AIN#(0:149)_EF_READ_D@33)

Address: 7966

```
property ain34_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN34_BINARY (AIN#(0:254)_BINARY@34)

Address: 50068

```
property ain34_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@34)

Address: 9368

```
property ain34_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@34)

Address: 9668

```
property ain34_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@34)

Address: 9968

```
property ain34_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@34)

Address: 10268

```
property ain34_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@34)

Address: 10568

```
property ain34_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@34)

Address: 10868

```
property ain34_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@34)

Address: 11168

```
property ain34_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@34)

Address: 11468

```
property ain34_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@34)

Address: 11768

```
property ain34_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@34)

Address: 12068

```
property ain34_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN34_EF_INDEX (AIN#(0:149)_EF_INDEX@34)

Address: 9068

```
property ain34_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN34_EF_READ_A (AIN#(0:149)_EF_READ_A@34)

Address: 7068

```
property ain34_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN34_EF_READ_B (AIN#(0:149)_EF_READ_B@34)

Address: 7368

```
property ain34_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN34_EF_READ_C (AIN#(0:149)_EF_READ_C@34)

Address: 7668

```
property ain34_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN34_EF_READ_D (AIN#(0:149)_EF_READ_D@34)

Address: 7968

```
property ain35_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN35_BINARY (AIN#(0:254)_BINARY@35)

Address: 50070

```
property ain35_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@35)

Address: 9370

```
property ain35_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@35)

Address: 9670

```
property ain35_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@35)

Address: 9970

```
property ain35_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@35)

Address: 10270

```
property ain35_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@35)

Address: 10570

```
property ain35_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@35)

Address: 10870

```
property ain35_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@35)

Address: 11170

```
property ain35_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@35)

Address: 11470

```
property ain35_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@35)

Address: 11770

```
property ain35_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@35)

Address: 12070

```
property ain35_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN35_EF_INDEX (AIN#(0:149)_EF_INDEX@35)

Address: 9070

```
property ain35_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN35_EF_READ_A (AIN#(0:149)_EF_READ_A@35)

Address: 7070

```
property ain35_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN35_EF_READ_B (AIN#(0:149)_EF_READ_B@35)

Address: 7370

```
property ain35_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN35_EF_READ_C (AIN#(0:149)_EF_READ_C@35)

Address: 7670

```
property ain35_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN35_EF_READ_D (AIN#(0:149)_EF_READ_D@35)

Address: 7970

```
property ain36_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN36_BINARY (AIN#(0:254)_BINARY@36)

Address: 50072

```
property ain36_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@36)

Address: 9372

```
property ain36_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@36)

Address: 9672

```
property ain36_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@36)

Address: 9972

```
property ain36_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@36)

Address: 10272

```
property ain36_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@36)

Address: 10572

```
property ain36_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@36)

Address: 10872

```
property ain36_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@36)

Address: 11172

```
property ain36_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@36)

Address: 11472

```
property ain36_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@36)

Address: 11772

```
property ain36_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@36)

Address: 12072

```
property ain36_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN36_EF_INDEX (AIN#(0:149)_EF_INDEX@36)

Address: 9072

```
property ain36_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN36_EF_READ_A (AIN#(0:149)_EF_READ_A@36)

Address: 7072

```
property ain36_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN36_EF_READ_B (AIN#(0:149)_EF_READ_B@36)

Address: 7372

```
property ain36_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN36_EF_READ_C (AIN#(0:149)_EF_READ_C@36)

Address: 7672

```
property ain36_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN36_EF_READ_D (AIN#(0:149)_EF_READ_D@36)

Address: 7972

```
property ain37_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN37_BINARY (AIN#(0:254)_BINARY@37)

Address: 50074

```
property ain37_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@37)

Address: 9374

```
property ain37_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@37)

Address: 9674

```
property ain37_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@37)

Address: 9974

```
property ain37_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@37)

Address: 10274

```
property ain37_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@37)

Address: 10574

```
property ain37_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@37)

Address: 10874

```
property ain37_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@37)

Address: 11174

```
property ain37_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@37)

Address: 11474

```
property ain37_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@37)

Address: 11774

```
property ain37_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@37)

Address: 12074

```
property ain37_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN37_EF_INDEX (AIN#(0:149)_EF_INDEX@37)

Address: 9074

```
property ain37_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN37_EF_READ_A (AIN#(0:149)_EF_READ_A@37)

Address: 7074

```
property ain37_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN37_EF_READ_B (AIN#(0:149)_EF_READ_B@37)

Address: 7374

```
property ain37_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN37_EF_READ_C (AIN#(0:149)_EF_READ_C@37)

Address: 7674

```
property ain37_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN37_EF_READ_D (AIN#(0:149)_EF_READ_D@37)

Address: 7974

```
property ain38_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN38_BINARY (AIN#(0:254)_BINARY@38)

Address: 50076

```
property ain38_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@38)

Address: 9376

```
property ain38_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@38)

Address: 9676

```
property ain38_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@38)

Address: 9976

```
property ain38_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@38)

Address: 10276

```
property ain38_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@38)

Address: 10576

```
property ain38_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@38)

Address: 10876

```
property ain38_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@38)

Address: 11176

```
property ain38_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@38)

Address: 11476

```
property ain38_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@38)

Address: 11776

```
property ain38_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@38)

Address: 12076

```
property ain38_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN38_EF_INDEX (AIN#(0:149)_EF_INDEX@38)

Address: 9076

```
property ain38_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN38_EF_READ_A (AIN#(0:149)_EF_READ_A@38)

Address: 7076

```
property ain38_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN38_EF_READ_B (AIN#(0:149)_EF_READ_B@38)

Address: 7376

```
property ain38_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN38_EF_READ_C (AIN#(0:149)_EF_READ_C@38)

Address: 7676

```
property ain38_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN38_EF_READ_D (AIN#(0:149)_EF_READ_D@38)

Address: 7976

```
property ain39_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN39_BINARY (AIN#(0:254)_BINARY@39)

Address: 50078

```
property ain39_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@39)

Address: 9378

```
property ain39_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@39)

Address: 9678

```
property ain39_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@39)

Address: 9978

```
property ain39_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@39)

Address: 10278

```
property ain39_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@39)

Address: 10578

```
property ain39_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@39)

Address: 10878

```
property ain39_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@39)

Address: 11178

```
property ain39_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@39)

Address: 11478

```
property ain39_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@39)

Address: 11778

```
property ain39_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@39)

Address: 12078

```
property ain39_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN39_EF_INDEX (AIN#(0:149)_EF_INDEX@39)

Address: 9078

```
property ain39_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN39_EF_READ_A (AIN#(0:149)_EF_READ_A@39)

Address: 7078

```
property ain39_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN39_EF_READ_B (AIN#(0:149)_EF_READ_B@39)

Address: 7378

```
property ain39_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN39_EF_READ_C (AIN#(0:149)_EF_READ_C@39)

Address: 7678

```
property ain39_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN39_EF_READ_D (AIN#(0:149)_EF_READ_D@39)

Address: 7978

```
property ain3_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN3_BINARY (AIN#(0:254)_BINARY@3)

Address: 50006

```
property ain3_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@3)

Address: 9306

```
property ain3_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@3)

Address: 9606

```
property ain3_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@3)

Address: 9906

```
property ain3_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@3)

Address: 10206

```
property ain3_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@3)

Address: 10506

```
property ain3_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@3)

Address: 10806

```
property ain3_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@3)

Address: 11106

```
property ain3_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@3)

Address: 11406

```
property ain3_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@3)

Address: 11706

```
property ain3_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@3)

Address: 12006

```
property ain3_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN3_EF_INDEX (AIN#(0:149)_EF_INDEX@3)

Address: 9006

```
property ain3_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN3_EF_READ_A (AIN#(0:149)_EF_READ_A@3)

Address: 7006

```
property ain3_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN3_EF_READ_B (AIN#(0:149)_EF_READ_B@3)

Address: 7306

```
property ain3_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN3_EF_READ_C (AIN#(0:149)_EF_READ_C@3)

Address: 7606

```
property ain3_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN3_EF_READ_D (AIN#(0:149)_EF_READ_D@3)

Address: 7906

```
property ain3_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN3_NEGATIVE_CH (AIN#(0:3)_NEGATIVE_CH@3)

Address: 41003

```
property ain3_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN3_RANGE (AIN#(0:3)_RANGE@3)

Address: 40006

```
property ain3_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN3_RESOLUTION_INDEX (AIN#(0:3)_RESOLUTION_INDEX@3)

Address: 41503

```
property ain3_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN3_SETTLING_US (AIN#(0:3)_SETTLING_US@3)

Address: 42006

```
property ain4: float
```

Returns the voltage of the specified analog input.

Register: AIN4 (AIN#(0:11)@4)

Address: 8

```
property ain40_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN40_BINARY (AIN#(0:254)_BINARY@40)

Address: 50080

```
property ain40_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@40)

Address: 9380

```
property ain40_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@40)

Address: 9680

```
property ain40_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@40)

Address: 9980

```
property ain40_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@40)

Address: 10280

```
property ain40_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@40)

Address: 10580

```
property ain40_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@40)

Address: 10880

```
property ain40_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@40)

Address: 11180

```
property ain40_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@40)

Address: 11480

```
property ain40_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@40)

Address: 11780

```
property ain40_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@40)

Address: 12080

```
property ain40_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN40_EF_INDEX (AIN#(0:149)_EF_INDEX@40)

Address: 9080

```
property ain40_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN40_EF_READ_A (AIN#(0:149)_EF_READ_A@40)

Address: 7080

```
property ain40_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN40_EF_READ_B (AIN#(0:149)_EF_READ_B@40)

Address: 7380

```
property ain40_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN40_EF_READ_C (AIN#(0:149)_EF_READ_C@40)

Address: 7680

```
property ain40_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN40_EF_READ_D (AIN#(0:149)_EF_READ_D@40)

Address: 7980

```
property ain41_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN41_BINARY (AIN#(0:254)_BINARY@41)

Address: 50082

```
property ain41_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@41)

Address: 9382

```
property ain41_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@41)

Address: 9682

```
property ain41_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@41)

Address: 9982

```
property ain41_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@41)

Address: 10282

```
property ain41_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@41)

Address: 10582

```
property ain41_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@41)

Address: 10882

```
property ain41_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@41)

Address: 11182

```
property ain41_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@41)

Address: 11482

```
property ain41_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@41)

Address: 11782

```
property ain41_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@41)

Address: 12082

```
property ain41_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN41_EF_INDEX (AIN#(0:149)_EF_INDEX@41)

Address: 9082

```
property ain41_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN41_EF_READ_A (AIN#(0:149)_EF_READ_A@41)

Address: 7082

```
property ain41_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN41_EF_READ_B (AIN#(0:149)_EF_READ_B@41)

Address: 7382

```
property ain41_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN41_EF_READ_C (AIN#(0:149)_EF_READ_C@41)

Address: 7682

```
property ain41_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN41_EF_READ_D (AIN#(0:149)_EF_READ_D@41)

Address: 7982

```
property ain42_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN42_BINARY (AIN#(0:254)_BINARY@42)

Address: 50084

```
property ain42_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@42)

Address: 9384

```
property ain42_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@42)

Address: 9684

```
property ain42_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@42)

Address: 9984

```
property ain42_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@42)

Address: 10284

```
property ain42_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@42)

Address: 10584

```
property ain42_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@42)

Address: 10884

```
property ain42_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@42)

Address: 11184

```
property ain42_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@42)

Address: 11484

```
property ain42_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@42)

Address: 11784

```
property ain42_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@42)

Address: 12084

```
property ain42_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN42_EF_INDEX (AIN#(0:149)_EF_INDEX@42)

Address: 9084

```
property ain42_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN42_EF_READ_A (AIN#(0:149)_EF_READ_A@42)

Address: 7084

```
property ain42_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN42_EF_READ_B (AIN#(0:149)_EF_READ_B@42)

Address: 7384

```
property ain42_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN42_EF_READ_C (AIN#(0:149)_EF_READ_C@42)

Address: 7684

```
property ain42_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN42_EF_READ_D (AIN#(0:149)_EF_READ_D@42)

Address: 7984

```
property ain43_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN43_BINARY (AIN#(0:254)_BINARY@43)

Address: 50086

```
property ain43_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@43)

Address: 9386

```
property ain43_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@43)

Address: 9686

```
property ain43_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@43)

Address: 9986

```
property ain43_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@43)

Address: 10286

```
property ain43_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@43)

Address: 10586

```
property ain43_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@43)

Address: 10886

```
property ain43_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@43)

Address: 11186

```
property ain43_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@43)

Address: 11486

```
property ain43_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@43)

Address: 11786

```
property ain43_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@43)

Address: 12086

```
property ain43_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN43_EF_INDEX (AIN#(0:149)_EF_INDEX@43)

Address: 9086

```
property ain43_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN43_EF_READ_A (AIN#(0:149)_EF_READ_A@43)

Address: 7086

```
property ain43_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN43_EF_READ_B (AIN#(0:149)_EF_READ_B@43)

Address: 7386

```
property ain43_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN43_EF_READ_C (AIN#(0:149)_EF_READ_C@43)

Address: 7686

```
property ain43_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN43_EF_READ_D (AIN#(0:149)_EF_READ_D@43)

Address: 7986

```
property ain44_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN44_BINARY (AIN#(0:254)_BINARY@44)

Address: 50088

```
property ain44_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@44)

Address: 9388

```
property ain44_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@44)

Address: 9688

```
property ain44_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@44)

Address: 9988

```
property ain44_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@44)

Address: 10288

```
property ain44_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@44)

Address: 10588

```
property ain44_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@44)

Address: 10888

```
property ain44_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@44)

Address: 11188

```
property ain44_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@44)

Address: 11488

```
property ain44_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@44)

Address: 11788

```
property ain44_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@44)

Address: 12088

```
property ain44_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN44_EF_INDEX (AIN#(0:149)_EF_INDEX@44)

Address: 9088

```
property ain44_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN44_EF_READ_A (AIN#(0:149)_EF_READ_A@44)

Address: 7088

```
property ain44_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN44_EF_READ_B (AIN#(0:149)_EF_READ_B@44)

Address: 7388

```
property ain44_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN44_EF_READ_C (AIN#(0:149)_EF_READ_C@44)

Address: 7688

```
property ain44_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN44_EF_READ_D (AIN#(0:149)_EF_READ_D@44)

Address: 7988

```
property ain45_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN45_BINARY (AIN#(0:254)_BINARY@45)

Address: 50090

```
property ain45_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@45)

Address: 9390

```
property ain45_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@45)

Address: 9690

```
property ain45_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@45)

Address: 9990

```
property ain45_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@45)

Address: 10290

```
property ain45_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@45)

Address: 10590

```
property ain45_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@45)

Address: 10890

```
property ain45_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@45)

Address: 11190

```
property ain45_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@45)

Address: 11490

```
property ain45_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@45)

Address: 11790

```
property ain45_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@45)

Address: 12090

```
property ain45_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN45_EF_INDEX (AIN#(0:149)_EF_INDEX@45)

Address: 9090

```
property ain45_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN45_EF_READ_A (AIN#(0:149)_EF_READ_A@45)

Address: 7090

```
property ain45_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN45_EF_READ_B (AIN#(0:149)_EF_READ_B@45)

Address: 7390

```
property ain45_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN45_EF_READ_C (AIN#(0:149)_EF_READ_C@45)

Address: 7690

```
property ain45_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN45_EF_READ_D (AIN#(0:149)_EF_READ_D@45)

Address: 7990

```
property ain46_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN46_BINARY (AIN#(0:254)_BINARY@46)

Address: 50092

```
property ain46_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@46)

Address: 9392

```
property ain46_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@46)

Address: 9692

```
property ain46_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@46)

Address: 9992

```
property ain46_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@46)

Address: 10292

```
property ain46_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@46)

Address: 10592

```
property ain46_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@46)

Address: 10892

```
property ain46_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@46)

Address: 11192

```
property ain46_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@46)

Address: 11492

```
property ain46_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@46)

Address: 11792

```
property ain46_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@46)

Address: 12092

```
property ain46_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN46_EF_INDEX (AIN#(0:149)_EF_INDEX@46)

Address: 9092

```
property ain46_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN46_EF_READ_A (AIN#(0:149)_EF_READ_A@46)

Address: 7092

```
property ain46_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN46_EF_READ_B (AIN#(0:149)_EF_READ_B@46)

Address: 7392

```
property ain46_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN46_EF_READ_C (AIN#(0:149)_EF_READ_C@46)

Address: 7692

```
property ain46_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN46_EF_READ_D (AIN#(0:149)_EF_READ_D@46)

Address: 7992

```
property ain47_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN47_BINARY (AIN#(0:254)_BINARY@47)

Address: 50094

```
property ain47_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@47)

Address: 9394

```
property ain47_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@47)

Address: 9694

```
property ain47_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@47)

Address: 9994

```
property ain47_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@47)

Address: 10294

```
property ain47_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@47)

Address: 10594

```
property ain47_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@47)

Address: 10894

```
property ain47_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@47)

Address: 11194

```
property ain47_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@47)

Address: 11494

```
property ain47_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@47)

Address: 11794

```
property ain47_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@47)

Address: 12094

```
property ain47_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN47_EF_INDEX (AIN#(0:149)_EF_INDEX@47)

Address: 9094

```
property ain47_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN47_EF_READ_A (AIN#(0:149)_EF_READ_A@47)

Address: 7094

```
property ain47_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN47_EF_READ_B (AIN#(0:149)_EF_READ_B@47)

Address: 7394

```
property ain47_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN47_EF_READ_C (AIN#(0:149)_EF_READ_C@47)

Address: 7694

```
property ain47_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN47_EF_READ_D (AIN#(0:149)_EF_READ_D@47)

Address: 7994

```
property ain48_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN48_BINARY (AIN#(0:254)_BINARY@48)

Address: 50096

```
property ain48_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@48)

Address: 9396

```
property ain48_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@48)

Address: 9696

```
property ain48_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@48)

Address: 9996

```
property ain48_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@48)

Address: 10296

```
property ain48_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@48)

Address: 10596

```
property ain48_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@48)

Address: 10896

```
property ain48_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@48)

Address: 11196

```
property ain48_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@48)

Address: 11496

```
property ain48_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@48)

Address: 11796

```
property ain48_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@48)

Address: 12096

```
property ain48_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN48_EF_INDEX (AIN#(0:149)_EF_INDEX@48)

Address: 9096

```
property ain48_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN48_EF_READ_A (AIN#(0:149)_EF_READ_A@48)

Address: 7096

```
property ain48_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN48_EF_READ_B (AIN#(0:149)_EF_READ_B@48)

Address: 7396

```
property ain48_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN48_EF_READ_C (AIN#(0:149)_EF_READ_C@48)

Address: 7696

```
property ain48_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN48_EF_READ_D (AIN#(0:149)_EF_READ_D@48)

Address: 7996

```
property ain49_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN49_BINARY (AIN#(0:254)_BINARY@49)

Address: 50098

```
property ain49_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@49)

Address: 9398

```
property ain49_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@49)

Address: 9698

```
property ain49_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@49)

Address: 9998

```
property ain49_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@49)

Address: 10298

```
property ain49_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@49)

Address: 10598

```
property ain49_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@49)

Address: 10898

```
property ain49_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@49)

Address: 11198

```
property ain49_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@49)

Address: 11498

```
property ain49_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@49)

Address: 11798

```
property ain49_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@49)

Address: 12098

```
property ain49_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN49_EF_INDEX (AIN#(0:149)_EF_INDEX@49)

Address: 9098

```
property ain49_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN49_EF_READ_A (AIN#(0:149)_EF_READ_A@49)

Address: 7098

```
property ain49_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN49_EF_READ_B (AIN#(0:149)_EF_READ_B@49)

Address: 7398

```
property ain49_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN49_EF_READ_C (AIN#(0:149)_EF_READ_C@49)

Address: 7698

```
property ain49_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN49_EF_READ_D (AIN#(0:149)_EF_READ_D@49)

Address: 7998

```
property ain4_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN4_BINARY (AIN#(0:254)_BINARY@4)

Address: 50008

```
property ain4_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@4)

Address: 9308

```
property ain4_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@4)

Address: 9608

```
property ain4_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@4)

Address: 9908

```
property ain4_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@4)

Address: 10208

```
property ain4_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@4)

Address: 10508

```
property ain4_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@4)

Address: 10808

```
property ain4_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@4)

Address: 11108

```
property ain4_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@4)

Address: 11408

```
property ain4_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@4)

Address: 11708

```
property ain4_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@4)

Address: 12008

```
property ain4_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN4_EF_INDEX (AIN#(0:149)_EF_INDEX@4)

Address: 9008

```
property ain4_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN4_EF_READ_A (AIN#(0:149)_EF_READ_A@4)

Address: 7008

```
property ain4_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN4_EF_READ_B (AIN#(0:149)_EF_READ_B@4)

Address: 7308

```
property ain4_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN4_EF_READ_C (AIN#(0:149)_EF_READ_C@4)

Address: 7608

```
property ain4_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN4_EF_READ_D (AIN#(0:149)_EF_READ_D@4)

Address: 7908

```
property ain5: float
```

Returns the voltage of the specified analog input.

Register: AIN5 (AIN#(0:11)@5)

Address: 10

```
property ain50_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN50_BINARY (AIN#(0:254)_BINARY@50)

Address: 50100

```
property ain50_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@50)

Address: 9400

```
property ain50_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@50)

Address: 9700

```
property ain50_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@50)

Address: 10000

```
property ain50_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@50)

Address: 10300

```
property ain50_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@50)

Address: 10600

```
property ain50_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@50)

Address: 10900

```
property ain50_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@50)

Address: 11200

```
property ain50_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@50)

Address: 11500

```
property ain50_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@50)

Address: 11800

```
property ain50_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@50)

Address: 12100

```
property ain50_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN50_EF_INDEX (AIN#(0:149)_EF_INDEX@50)

Address: 9100

```
property ain50_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN50_EF_READ_A (AIN#(0:149)_EF_READ_A@50)

Address: 7100

```
property ain50_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN50_EF_READ_B (AIN#(0:149)_EF_READ_B@50)

Address: 7400

```
property ain50_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN50_EF_READ_C (AIN#(0:149)_EF_READ_C@50)

Address: 7700

```
property ain50_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN50_EF_READ_D (AIN#(0:149)_EF_READ_D@50)

Address: 8000

```
property ain51_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN51_BINARY (AIN#(0:254)_BINARY@51)

Address: 50102

```
property ain51_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@51)

Address: 9402

```
property ain51_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@51)

Address: 9702

```
property ain51_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@51)

Address: 10002

```
property ain51_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@51)

Address: 10302

```
property ain51_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@51)

Address: 10602

```
property ain51_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@51)

Address: 10902

```
property ain51_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@51)

Address: 11202

```
property ain51_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@51)

Address: 11502

```
property ain51_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@51)

Address: 11802

```
property ain51_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@51)

Address: 12102

```
property ain51_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN51_EF_INDEX (AIN#(0:149)_EF_INDEX@51)

Address: 9102

```
property ain51_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN51_EF_READ_A (AIN#(0:149)_EF_READ_A@51)

Address: 7102

```
property ain51_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN51_EF_READ_B (AIN#(0:149)_EF_READ_B@51)

Address: 7402

```
property ain51_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN51_EF_READ_C (AIN#(0:149)_EF_READ_C@51)

Address: 7702

```
property ain51_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN51_EF_READ_D (AIN#(0:149)_EF_READ_D@51)

Address: 8002

```
property ain52_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN52_BINARY (AIN#(0:254)_BINARY@52)

Address: 50104

```
property ain52_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@52)

Address: 9404

```
property ain52_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@52)

Address: 9704

```
property ain52_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@52)

Address: 10004

```
property ain52_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@52)

Address: 10304

```
property ain52_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@52)

Address: 10604

```
property ain52_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@52)

Address: 10904

```
property ain52_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@52)

Address: 11204

```
property ain52_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@52)

Address: 11504

```
property ain52_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@52)

Address: 11804

```
property ain52_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@52)

Address: 12104

```
property ain52_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN52_EF_INDEX (AIN#(0:149)_EF_INDEX@52)

Address: 9104

```
property ain52_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN52_EF_READ_A (AIN#(0:149)_EF_READ_A@52)

Address: 7104

```
property ain52_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN52_EF_READ_B (AIN#(0:149)_EF_READ_B@52)

Address: 7404

```
property ain52_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN52_EF_READ_C (AIN#(0:149)_EF_READ_C@52)

Address: 7704

```
property ain52_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN52_EF_READ_D (AIN#(0:149)_EF_READ_D@52)

Address: 8004

```
property ain53_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN53_BINARY (AIN#(0:254)_BINARY@53)

Address: 50106

```
property ain53_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@53)

Address: 9406

```
property ain53_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@53)

Address: 9706

```
property ain53_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@53)

Address: 10006

```
property ain53_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@53)

Address: 10306

```
property ain53_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@53)

Address: 10606

```
property ain53_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@53)

Address: 10906

```
property ain53_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@53)

Address: 11206

```
property ain53_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@53)

Address: 11506

```
property ain53_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@53)

Address: 11806

```
property ain53_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@53)

Address: 12106

```
property ain53_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN53_EF_INDEX (AIN#(0:149)_EF_INDEX@53)

Address: 9106

```
property ain53_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN53_EF_READ_A (AIN#(0:149)_EF_READ_A@53)

Address: 7106

```
property ain53_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN53_EF_READ_B (AIN#(0:149)_EF_READ_B@53)

Address: 7406

```
property ain53_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN53_EF_READ_C (AIN#(0:149)_EF_READ_C@53)

Address: 7706

```
property ain53_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN53_EF_READ_D (AIN#(0:149)_EF_READ_D@53)

Address: 8006

```
property ain54_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN54_BINARY (AIN#(0:254)_BINARY@54)

Address: 50108

```
property ain54_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@54)

Address: 9408

```
property ain54_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@54)

Address: 9708

```
property ain54_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@54)

Address: 10008

```
property ain54_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@54)

Address: 10308

```
property ain54_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@54)

Address: 10608

```
property ain54_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@54)

Address: 10908

```
property ain54_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@54)

Address: 11208

```
property ain54_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@54)

Address: 11508

```
property ain54_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@54)

Address: 11808

```
property ain54_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@54)

Address: 12108

```
property ain54_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN54_EF_INDEX (AIN#(0:149)_EF_INDEX@54)

Address: 9108

```
property ain54_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN54_EF_READ_A (AIN#(0:149)_EF_READ_A@54)

Address: 7108

```
property ain54_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN54_EF_READ_B (AIN#(0:149)_EF_READ_B@54)

Address: 7408

```
property ain54_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN54_EF_READ_C (AIN#(0:149)_EF_READ_C@54)

Address: 7708

```
property ain54_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN54_EF_READ_D (AIN#(0:149)_EF_READ_D@54)

Address: 8008

```
property ain55_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN55_BINARY (AIN#(0:254)_BINARY@55)

Address: 50110

```
property ain55_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@55)

Address: 9410

```
property ain55_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@55)

Address: 9710

```
property ain55_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@55)

Address: 10010

```
property ain55_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@55)

Address: 10310

```
property ain55_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@55)

Address: 10610

```
property ain55_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@55)

Address: 10910

```
property ain55_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@55)

Address: 11210

```
property ain55_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@55)

Address: 11510

```
property ain55_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@55)

Address: 11810

```
property ain55_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@55)

Address: 12110

```
property ain55_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN55_EF_INDEX (AIN#(0:149)_EF_INDEX@55)

Address: 9110

```
property ain55_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN55_EF_READ_A (AIN#(0:149)_EF_READ_A@55)

Address: 7110

```
property ain55_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN55_EF_READ_B (AIN#(0:149)_EF_READ_B@55)

Address: 7410

```
property ain55_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN55_EF_READ_C (AIN#(0:149)_EF_READ_C@55)

Address: 7710

```
property ain55_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN55_EF_READ_D (AIN#(0:149)_EF_READ_D@55)

Address: 8010

```
property ain56_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN56_BINARY (AIN#(0:254)_BINARY@56)

Address: 50112

```
property ain56_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@56)

Address: 9412

```
property ain56_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@56)

Address: 9712

```
property ain56_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@56)

Address: 10012

```
property ain56_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@56)

Address: 10312

```
property ain56_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@56)

Address: 10612

```
property ain56_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@56)

Address: 10912

```
property ain56_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@56)

Address: 11212

```
property ain56_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@56)

Address: 11512

```
property ain56_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@56)

Address: 11812

```
property ain56_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@56)

Address: 12112

```
property ain56_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN56_EF_INDEX (AIN#(0:149)_EF_INDEX@56)

Address: 9112

```
property ain56_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN56_EF_READ_A (AIN#(0:149)_EF_READ_A@56)

Address: 7112

```
property ain56_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN56_EF_READ_B (AIN#(0:149)_EF_READ_B@56)

Address: 7412

```
property ain56_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN56_EF_READ_C (AIN#(0:149)_EF_READ_C@56)

Address: 7712

```
property ain56_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN56_EF_READ_D (AIN#(0:149)_EF_READ_D@56)

Address: 8012

```
property ain57_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN57_BINARY (AIN#(0:254)_BINARY@57)

Address: 50114

```
property ain57_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@57)

Address: 9414

```
property ain57_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@57)

Address: 9714

```
property ain57_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@57)

Address: 10014

```
property ain57_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@57)

Address: 10314

```
property ain57_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@57)

Address: 10614

```
property ain57_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@57)

Address: 10914

```
property ain57_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@57)

Address: 11214

```
property ain57_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@57)

Address: 11514

```
property ain57_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@57)

Address: 11814

```
property ain57_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@57)

Address: 12114

```
property ain57_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN57_EF_INDEX (AIN#(0:149)_EF_INDEX@57)

Address: 9114

```
property ain57_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN57_EF_READ_A (AIN#(0:149)_EF_READ_A@57)

Address: 7114

```
property ain57_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN57_EF_READ_B (AIN#(0:149)_EF_READ_B@57)

Address: 7414

```
property ain57_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN57_EF_READ_C (AIN#(0:149)_EF_READ_C@57)

Address: 7714

```
property ain57_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN57_EF_READ_D (AIN#(0:149)_EF_READ_D@57)

Address: 8014

```
property ain58_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN58_BINARY (AIN#(0:254)_BINARY@58)

Address: 50116

```
property ain58_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@58)

Address: 9416

```
property ain58_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@58)

Address: 9716

```
property ain58_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@58)

Address: 10016

```
property ain58_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@58)

Address: 10316

```
property ain58_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@58)

Address: 10616

```
property ain58_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@58)

Address: 10916

```
property ain58_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@58)

Address: 11216

```
property ain58_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@58)

Address: 11516

```
property ain58_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@58)

Address: 11816

```
property ain58_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@58)

Address: 12116

```
property ain58_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN58_EF_INDEX (AIN#(0:149)_EF_INDEX@58)

Address: 9116

```
property ain58_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN58_EF_READ_A (AIN#(0:149)_EF_READ_A@58)

Address: 7116

```
property ain58_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN58_EF_READ_B (AIN#(0:149)_EF_READ_B@58)

Address: 7416

```
property ain58_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN58_EF_READ_C (AIN#(0:149)_EF_READ_C@58)

Address: 7716

```
property ain58_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN58_EF_READ_D (AIN#(0:149)_EF_READ_D@58)

Address: 8016

```
property ain59_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN59_BINARY (AIN#(0:254)_BINARY@59)

Address: 50118

```
property ain59_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@59)

Address: 9418

```
property ain59_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@59)

Address: 9718

```
property ain59_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@59)

Address: 10018

```
property ain59_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@59)

Address: 10318

```
property ain59_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@59)

Address: 10618

```
property ain59_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@59)

Address: 10918

```
property ain59_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@59)

Address: 11218

```
property ain59_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@59)

Address: 11518

```
property ain59_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@59)

Address: 11818

```
property ain59_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@59)

Address: 12118

```
property ain59_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN59_EF_INDEX (AIN#(0:149)_EF_INDEX@59)

Address: 9118

```
property ain59_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN59_EF_READ_A (AIN#(0:149)_EF_READ_A@59)

Address: 7118

```
property ain59_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN59_EF_READ_B (AIN#(0:149)_EF_READ_B@59)

Address: 7418

```
property ain59_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN59_EF_READ_C (AIN#(0:149)_EF_READ_C@59)

Address: 7718

```
property ain59_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN59_EF_READ_D (AIN#(0:149)_EF_READ_D@59)

Address: 8018

```
property ain5_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN5_BINARY (AIN#(0:254)_BINARY@5)

Address: 50010

```
property ain5_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@5)

Address: 9310

```
property ain5_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@5)

Address: 9610

```
property ain5_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@5)

Address: 9910

```
property ain5_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@5)

Address: 10210

```
property ain5_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@5)

Address: 10510

```
property ain5_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@5)

Address: 10810

```
property ain5_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@5)

Address: 11110

```
property ain5_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@5)

Address: 11410

```
property ain5_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@5)

Address: 11710

```
property ain5_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@5)

Address: 12010

```
property ain5_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN5_EF_INDEX (AIN#(0:149)_EF_INDEX@5)

Address: 9010

```
property ain5_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN5_EF_READ_A (AIN#(0:149)_EF_READ_A@5)

Address: 7010

```
property ain5_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN5_EF_READ_B (AIN#(0:149)_EF_READ_B@5)

Address: 7310

```
property ain5_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN5_EF_READ_C (AIN#(0:149)_EF_READ_C@5)

Address: 7610

```
property ain5_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN5_EF_READ_D (AIN#(0:149)_EF_READ_D@5)

Address: 7910

```
property ain6: float
```

Returns the voltage of the specified analog input.

Register: AIN6 (AIN#(0:11)@6)

Address: 12

```
property ain60_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN60_BINARY (AIN#(0:254)_BINARY@60)

Address: 50120

```
property ain60_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@60)

Address: 9420

```
property ain60_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@60)

Address: 9720

```
property ain60_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@60)

Address: 10020

```
property ain60_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@60)

Address: 10320

```
property ain60_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@60)

Address: 10620

```
property ain60_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@60)

Address: 10920

```
property ain60_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@60)

Address: 11220

```
property ain60_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@60)

Address: 11520

```
property ain60_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@60)

Address: 11820

```
property ain60_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@60)

Address: 12120

```
property ain60_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN60_EF_INDEX (AIN#(0:149)_EF_INDEX@60)

Address: 9120

```
property ain60_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN60_EF_READ_A (AIN#(0:149)_EF_READ_A@60)

Address: 7120

```
property ain60_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN60_EF_READ_B (AIN#(0:149)_EF_READ_B@60)

Address: 7420

```
property ain60_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN60_EF_READ_C (AIN#(0:149)_EF_READ_C@60)

Address: 7720

```
property ain60_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN60_EF_READ_D (AIN#(0:149)_EF_READ_D@60)

Address: 8020

```
property ain61_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN61_BINARY (AIN#(0:254)_BINARY@61)

Address: 50122

```
property ain61_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@61)

Address: 9422

```
property ain61_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@61)

Address: 9722

```
property ain61_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@61)

Address: 10022

```
property ain61_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@61)

Address: 10322

```
property ain61_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@61)

Address: 10622

```
property ain61_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@61)

Address: 10922

```
property ain61_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@61)

Address: 11222

```
property ain61_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@61)

Address: 11522

```
property ain61_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@61)

Address: 11822

```
property ain61_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@61)

Address: 12122

```
property ain61_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN61_EF_INDEX (AIN#(0:149)_EF_INDEX@61)

Address: 9122

```
property ain61_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN61_EF_READ_A (AIN#(0:149)_EF_READ_A@61)

Address: 7122

```
property ain61_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN61_EF_READ_B (AIN#(0:149)_EF_READ_B@61)

Address: 7422

```
property ain61_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN61_EF_READ_C (AIN#(0:149)_EF_READ_C@61)

Address: 7722

```
property ain61_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN61_EF_READ_D (AIN#(0:149)_EF_READ_D@61)

Address: 8022

```
property ain62_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN62_BINARY (AIN#(0:254)_BINARY@62)

Address: 50124

```
property ain62_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@62)

Address: 9424

```
property ain62_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@62)

Address: 9724

```
property ain62_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@62)

Address: 10024

```
property ain62_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@62)

Address: 10324

```
property ain62_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@62)

Address: 10624

```
property ain62_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@62)

Address: 10924

```
property ain62_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@62)

Address: 11224

```
property ain62_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@62)

Address: 11524

```
property ain62_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@62)

Address: 11824

```
property ain62_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@62)

Address: 12124

```
property ain62_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN62_EF_INDEX (AIN#(0:149)_EF_INDEX@62)

Address: 9124

```
property ain62_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN62_EF_READ_A (AIN#(0:149)_EF_READ_A@62)

Address: 7124

```
property ain62_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN62_EF_READ_B (AIN#(0:149)_EF_READ_B@62)

Address: 7424

```
property ain62_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN62_EF_READ_C (AIN#(0:149)_EF_READ_C@62)

Address: 7724

```
property ain62_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN62_EF_READ_D (AIN#(0:149)_EF_READ_D@62)

Address: 8024

```
property ain63_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN63_BINARY (AIN#(0:254)_BINARY@63)

Address: 50126

```
property ain63_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@63)

Address: 9426

```
property ain63_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@63)

Address: 9726

```
property ain63_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@63)

Address: 10026

```
property ain63_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@63)

Address: 10326

```
property ain63_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@63)

Address: 10626

```
property ain63_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@63)

Address: 10926

```
property ain63_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@63)

Address: 11226

```
property ain63_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@63)

Address: 11526

```
property ain63_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@63)

Address: 11826

```
property ain63_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@63)

Address: 12126

```
property ain63_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN63_EF_INDEX (AIN#(0:149)_EF_INDEX@63)

Address: 9126

```
property ain63_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN63_EF_READ_A (AIN#(0:149)_EF_READ_A@63)

Address: 7126

```
property ain63_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN63_EF_READ_B (AIN#(0:149)_EF_READ_B@63)

Address: 7426

```
property ain63_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN63_EF_READ_C (AIN#(0:149)_EF_READ_C@63)

Address: 7726

```
property ain63_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN63_EF_READ_D (AIN#(0:149)_EF_READ_D@63)

Address: 8026

```
property ain64_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN64_BINARY (AIN#(0:254)_BINARY@64)

Address: 50128

```
property ain64_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@64)

Address: 9428

```
property ain64_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@64)

Address: 9728

```
property ain64_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@64)

Address: 10028

```
property ain64_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@64)

Address: 10328

```
property ain64_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@64)

Address: 10628

```
property ain64_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@64)

Address: 10928

```
property ain64_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@64)

Address: 11228

```
property ain64_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@64)

Address: 11528

```
property ain64_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@64)

Address: 11828

```
property ain64_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@64)

Address: 12128

```
property ain64_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN64_EF_INDEX (AIN#(0:149)_EF_INDEX@64)

Address: 9128

```
property ain64_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN64_EF_READ_A (AIN#(0:149)_EF_READ_A@64)

Address: 7128

```
property ain64_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN64_EF_READ_B (AIN#(0:149)_EF_READ_B@64)

Address: 7428

```
property ain64_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN64_EF_READ_C (AIN#(0:149)_EF_READ_C@64)

Address: 7728

```
property ain64_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN64_EF_READ_D (AIN#(0:149)_EF_READ_D@64)

Address: 8028

```
property ain65_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN65_BINARY (AIN#(0:254)_BINARY@65)

Address: 50130

```
property ain65_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@65)

Address: 9430

```
property ain65_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@65)

Address: 9730

```
property ain65_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@65)

Address: 10030

```
property ain65_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@65)

Address: 10330

```
property ain65_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@65)

Address: 10630

```
property ain65_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@65)

Address: 10930

```
property ain65_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@65)

Address: 11230

```
property ain65_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@65)

Address: 11530

```
property ain65_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@65)

Address: 11830

```
property ain65_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@65)

Address: 12130

```
property ain65_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN65_EF_INDEX (AIN#(0:149)_EF_INDEX@65)

Address: 9130

```
property ain65_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN65_EF_READ_A (AIN#(0:149)_EF_READ_A@65)

Address: 7130

```
property ain65_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN65_EF_READ_B (AIN#(0:149)_EF_READ_B@65)

Address: 7430

```
property ain65_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN65_EF_READ_C (AIN#(0:149)_EF_READ_C@65)

Address: 7730

```
property ain65_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN65_EF_READ_D (AIN#(0:149)_EF_READ_D@65)

Address: 8030

```
property ain66_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN66_BINARY (AIN#(0:254)_BINARY@66)

Address: 50132

```
property ain66_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@66)

Address: 9432

```
property ain66_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@66)

Address: 9732

```
property ain66_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@66)

Address: 10032

```
property ain66_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@66)

Address: 10332

```
property ain66_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@66)

Address: 10632

```
property ain66_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@66)

Address: 10932

```
property ain66_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@66)

Address: 11232

```
property ain66_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@66)

Address: 11532

```
property ain66_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@66)

Address: 11832

```
property ain66_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@66)

Address: 12132

```
property ain66_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN66_EF_INDEX (AIN#(0:149)_EF_INDEX@66)

Address: 9132

```
property ain66_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN66_EF_READ_A (AIN#(0:149)_EF_READ_A@66)

Address: 7132

```
property ain66_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN66_EF_READ_B (AIN#(0:149)_EF_READ_B@66)

Address: 7432

```
property ain66_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN66_EF_READ_C (AIN#(0:149)_EF_READ_C@66)

Address: 7732

```
property ain66_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN66_EF_READ_D (AIN#(0:149)_EF_READ_D@66)

Address: 8032

```
property ain67_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN67_BINARY (AIN#(0:254)_BINARY@67)

Address: 50134

```
property ain67_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@67)

Address: 9434

```
property ain67_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@67)

Address: 9734

```
property ain67_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@67)

Address: 10034

```
property ain67_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@67)

Address: 10334

```
property ain67_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@67)

Address: 10634

```
property ain67_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@67)

Address: 10934

```
property ain67_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@67)

Address: 11234

```
property ain67_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@67)

Address: 11534

```
property ain67_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@67)

Address: 11834

```
property ain67_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@67)

Address: 12134

```
property ain67_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN67_EF_INDEX (AIN#(0:149)_EF_INDEX@67)

Address: 9134

```
property ain67_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN67_EF_READ_A (AIN#(0:149)_EF_READ_A@67)

Address: 7134

```
property ain67_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN67_EF_READ_B (AIN#(0:149)_EF_READ_B@67)

Address: 7434

```
property ain67_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN67_EF_READ_C (AIN#(0:149)_EF_READ_C@67)

Address: 7734

```
property ain67_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN67_EF_READ_D (AIN#(0:149)_EF_READ_D@67)

Address: 8034

```
property ain68_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN68_BINARY (AIN#(0:254)_BINARY@68)

Address: 50136

```
property ain68_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@68)

Address: 9436

```
property ain68_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@68)

Address: 9736

```
property ain68_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@68)

Address: 10036

```
property ain68_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@68)

Address: 10336

```
property ain68_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@68)

Address: 10636

```
property ain68_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@68)

Address: 10936

```
property ain68_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@68)

Address: 11236

```
property ain68_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@68)

Address: 11536

```
property ain68_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@68)

Address: 11836

```
property ain68_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@68)

Address: 12136

```
property ain68_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN68_EF_INDEX (AIN#(0:149)_EF_INDEX@68)

Address: 9136

```
property ain68_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN68_EF_READ_A (AIN#(0:149)_EF_READ_A@68)

Address: 7136

```
property ain68_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN68_EF_READ_B (AIN#(0:149)_EF_READ_B@68)

Address: 7436

```
property ain68_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN68_EF_READ_C (AIN#(0:149)_EF_READ_C@68)

Address: 7736

```
property ain68_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN68_EF_READ_D (AIN#(0:149)_EF_READ_D@68)

Address: 8036

```
property ain69_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN69_BINARY (AIN#(0:254)_BINARY@69)

Address: 50138

```
property ain69_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@69)

Address: 9438

```
property ain69_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@69)

Address: 9738

```
property ain69_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@69)

Address: 10038

```
property ain69_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@69)

Address: 10338

```
property ain69_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@69)

Address: 10638

```
property ain69_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@69)

Address: 10938

```
property ain69_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@69)

Address: 11238

```
property ain69_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@69)

Address: 11538

```
property ain69_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@69)

Address: 11838

```
property ain69_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@69)

Address: 12138

```
property ain69_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN69_EF_INDEX (AIN#(0:149)_EF_INDEX@69)

Address: 9138

```
property ain69_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN69_EF_READ_A (AIN#(0:149)_EF_READ_A@69)

Address: 7138

```
property ain69_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN69_EF_READ_B (AIN#(0:149)_EF_READ_B@69)

Address: 7438

```
property ain69_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN69_EF_READ_C (AIN#(0:149)_EF_READ_C@69)

Address: 7738

```
property ain69_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN69_EF_READ_D (AIN#(0:149)_EF_READ_D@69)

Address: 8038

```
property ain6_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN6_BINARY (AIN#(0:254)_BINARY@6)

Address: 50012

```
property ain6_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@6)

Address: 9312

```
property ain6_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@6)

Address: 9612

```
property ain6_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@6)

Address: 9912

```
property ain6_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@6)

Address: 10212

```
property ain6_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@6)

Address: 10512

```
property ain6_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@6)

Address: 10812

```
property ain6_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@6)

Address: 11112

```
property ain6_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@6)

Address: 11412

```
property ain6_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@6)

Address: 11712

```
property ain6_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@6)

Address: 12012

```
property ain6_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN6_EF_INDEX (AIN#(0:149)_EF_INDEX@6)

Address: 9012

```
property ain6_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN6_EF_READ_A (AIN#(0:149)_EF_READ_A@6)

Address: 7012

```
property ain6_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN6_EF_READ_B (AIN#(0:149)_EF_READ_B@6)

Address: 7312

```
property ain6_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN6_EF_READ_C (AIN#(0:149)_EF_READ_C@6)

Address: 7612

```
property ain6_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN6_EF_READ_D (AIN#(0:149)_EF_READ_D@6)

Address: 7912

```
property ain7: float
```

Returns the voltage of the specified analog input.

Register: AIN7 (AIN#(0:11)@7)

Address: 14

```
property ain70_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN70_BINARY (AIN#(0:254)_BINARY@70)

Address: 50140

```
property ain70_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@70)

Address: 9440

```
property ain70_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@70)

Address: 9740

```
property ain70_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@70)

Address: 10040

```
property ain70_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@70)

Address: 10340

```
property ain70_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@70)

Address: 10640

```
property ain70_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@70)

Address: 10940

```
property ain70_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@70)

Address: 11240

```
property ain70_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@70)

Address: 11540

```
property ain70_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@70)

Address: 11840

```
property ain70_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@70)

Address: 12140

```
property ain70_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN70_EF_INDEX (AIN#(0:149)_EF_INDEX@70)

Address: 9140

```
property ain70_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN70_EF_READ_A (AIN#(0:149)_EF_READ_A@70)

Address: 7140

```
property ain70_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN70_EF_READ_B (AIN#(0:149)_EF_READ_B@70)

Address: 7440

```
property ain70_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN70_EF_READ_C (AIN#(0:149)_EF_READ_C@70)

Address: 7740

```
property ain70_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN70_EF_READ_D (AIN#(0:149)_EF_READ_D@70)

Address: 8040

```
property ain71_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN71_BINARY (AIN#(0:254)_BINARY@71)

Address: 50142

```
property ain71_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@71)

Address: 9442

```
property ain71_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@71)

Address: 9742

```
property ain71_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@71)

Address: 10042

```
property ain71_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@71)

Address: 10342

```
property ain71_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@71)

Address: 10642

```
property ain71_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@71)

Address: 10942

```
property ain71_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@71)

Address: 11242

```
property ain71_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@71)

Address: 11542

```
property ain71_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@71)

Address: 11842

```
property ain71_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@71)

Address: 12142

```
property ain71_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN71_EF_INDEX (AIN#(0:149)_EF_INDEX@71)

Address: 9142

```
property ain71_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN71_EF_READ_A (AIN#(0:149)_EF_READ_A@71)

Address: 7142

```
property ain71_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN71_EF_READ_B (AIN#(0:149)_EF_READ_B@71)

Address: 7442

```
property ain71_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN71_EF_READ_C (AIN#(0:149)_EF_READ_C@71)

Address: 7742

```
property ain71_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN71_EF_READ_D (AIN#(0:149)_EF_READ_D@71)

Address: 8042

```
property ain72_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN72_BINARY (AIN#(0:254)_BINARY@72)

Address: 50144

```
property ain72_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@72)

Address: 9444

```
property ain72_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@72)

Address: 9744

```
property ain72_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@72)

Address: 10044

```
property ain72_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@72)

Address: 10344

```
property ain72_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@72)

Address: 10644

```
property ain72_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@72)

Address: 10944

```
property ain72_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@72)

Address: 11244

```
property ain72_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@72)

Address: 11544

```
property ain72_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@72)

Address: 11844

```
property ain72_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@72)

Address: 12144

```
property ain72_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN72_EF_INDEX (AIN#(0:149)_EF_INDEX@72)

Address: 9144

```
property ain72_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN72_EF_READ_A (AIN#(0:149)_EF_READ_A@72)

Address: 7144

```
property ain72_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN72_EF_READ_B (AIN#(0:149)_EF_READ_B@72)

Address: 7444

```
property ain72_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN72_EF_READ_C (AIN#(0:149)_EF_READ_C@72)

Address: 7744

```
property ain72_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN72_EF_READ_D (AIN#(0:149)_EF_READ_D@72)

Address: 8044

```
property ain73_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN73_BINARY (AIN#(0:254)_BINARY@73)

Address: 50146

```
property ain73_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@73)

Address: 9446

```
property ain73_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@73)

Address: 9746

```
property ain73_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@73)

Address: 10046

```
property ain73_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@73)

Address: 10346

```
property ain73_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@73)

Address: 10646

```
property ain73_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@73)

Address: 10946

```
property ain73_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@73)

Address: 11246

```
property ain73_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@73)

Address: 11546

```
property ain73_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@73)

Address: 11846

```
property ain73_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@73)

Address: 12146

```
property ain73_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN73_EF_INDEX (AIN#(0:149)_EF_INDEX@73)

Address: 9146

```
property ain73_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN73_EF_READ_A (AIN#(0:149)_EF_READ_A@73)

Address: 7146

```
property ain73_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN73_EF_READ_B (AIN#(0:149)_EF_READ_B@73)

Address: 7446

```
property ain73_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN73_EF_READ_C (AIN#(0:149)_EF_READ_C@73)

Address: 7746

```
property ain73_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN73_EF_READ_D (AIN#(0:149)_EF_READ_D@73)

Address: 8046

```
property ain74_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN74_BINARY (AIN#(0:254)_BINARY@74)

Address: 50148

```
property ain74_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@74)

Address: 9448

```
property ain74_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@74)

Address: 9748

```
property ain74_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@74)

Address: 10048

```
property ain74_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@74)

Address: 10348

```
property ain74_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@74)

Address: 10648

```
property ain74_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@74)

Address: 10948

```
property ain74_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@74)

Address: 11248

```
property ain74_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@74)

Address: 11548

```
property ain74_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@74)

Address: 11848

```
property ain74_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@74)

Address: 12148

```
property ain74_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN74_EF_INDEX (AIN#(0:149)_EF_INDEX@74)

Address: 9148

```
property ain74_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN74_EF_READ_A (AIN#(0:149)_EF_READ_A@74)

Address: 7148

```
property ain74_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN74_EF_READ_B (AIN#(0:149)_EF_READ_B@74)

Address: 7448

```
property ain74_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN74_EF_READ_C (AIN#(0:149)_EF_READ_C@74)

Address: 7748

```
property ain74_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN74_EF_READ_D (AIN#(0:149)_EF_READ_D@74)

Address: 8048

```
property ain75_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN75_BINARY (AIN#(0:254)_BINARY@75)

Address: 50150

```
property ain75_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@75)

Address: 9450

```
property ain75_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@75)

Address: 9750

```
property ain75_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@75)

Address: 10050

```
property ain75_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@75)

Address: 10350

```
property ain75_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@75)

Address: 10650

```
property ain75_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@75)

Address: 10950

```
property ain75_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@75)

Address: 11250

```
property ain75_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@75)

Address: 11550

```
property ain75_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@75)

Address: 11850

```
property ain75_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@75)

Address: 12150

```
property ain75_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN75_EF_INDEX (AIN#(0:149)_EF_INDEX@75)

Address: 9150

```
property ain75_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN75_EF_READ_A (AIN#(0:149)_EF_READ_A@75)

Address: 7150

```
property ain75_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN75_EF_READ_B (AIN#(0:149)_EF_READ_B@75)

Address: 7450

```
property ain75_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN75_EF_READ_C (AIN#(0:149)_EF_READ_C@75)

Address: 7750

```
property ain75_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN75_EF_READ_D (AIN#(0:149)_EF_READ_D@75)

Address: 8050

```
property ain76_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN76_BINARY (AIN#(0:254)_BINARY@76)

Address: 50152

```
property ain76_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@76)

Address: 9452

```
property ain76_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@76)

Address: 9752

```
property ain76_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@76)

Address: 10052

```
property ain76_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@76)

Address: 10352

```
property ain76_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@76)

Address: 10652

```
property ain76_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@76)

Address: 10952

```
property ain76_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@76)

Address: 11252

```
property ain76_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@76)

Address: 11552

```
property ain76_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@76)

Address: 11852

```
property ain76_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@76)

Address: 12152

```
property ain76_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN76_EF_INDEX (AIN#(0:149)_EF_INDEX@76)

Address: 9152

```
property ain76_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN76_EF_READ_A (AIN#(0:149)_EF_READ_A@76)

Address: 7152

```
property ain76_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN76_EF_READ_B (AIN#(0:149)_EF_READ_B@76)

Address: 7452

```
property ain76_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN76_EF_READ_C (AIN#(0:149)_EF_READ_C@76)

Address: 7752

```
property ain76_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN76_EF_READ_D (AIN#(0:149)_EF_READ_D@76)

Address: 8052

```
property ain77_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN77_BINARY (AIN#(0:254)_BINARY@77)

Address: 50154

```
property ain77_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@77)

Address: 9454

```
property ain77_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@77)

Address: 9754

```
property ain77_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@77)

Address: 10054

```
property ain77_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@77)

Address: 10354

```
property ain77_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@77)

Address: 10654

```
property ain77_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@77)

Address: 10954

```
property ain77_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@77)

Address: 11254

```
property ain77_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@77)

Address: 11554

```
property ain77_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@77)

Address: 11854

```
property ain77_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@77)

Address: 12154

```
property ain77_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN77_EF_INDEX (AIN#(0:149)_EF_INDEX@77)

Address: 9154

```
property ain77_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN77_EF_READ_A (AIN#(0:149)_EF_READ_A@77)

Address: 7154

```
property ain77_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN77_EF_READ_B (AIN#(0:149)_EF_READ_B@77)

Address: 7454

```
property ain77_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN77_EF_READ_C (AIN#(0:149)_EF_READ_C@77)

Address: 7754

```
property ain77_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN77_EF_READ_D (AIN#(0:149)_EF_READ_D@77)

Address: 8054

```
property ain78_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN78_BINARY (AIN#(0:254)_BINARY@78)

Address: 50156

```
property ain78_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@78)

Address: 9456

```
property ain78_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@78)

Address: 9756

```
property ain78_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@78)

Address: 10056

```
property ain78_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@78)

Address: 10356

```
property ain78_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@78)

Address: 10656

```
property ain78_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@78)

Address: 10956

```
property ain78_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@78)

Address: 11256

```
property ain78_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@78)

Address: 11556

```
property ain78_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@78)

Address: 11856

```
property ain78_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@78)

Address: 12156

```
property ain78_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN78_EF_INDEX (AIN#(0:149)_EF_INDEX@78)

Address: 9156

```
property ain78_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN78_EF_READ_A (AIN#(0:149)_EF_READ_A@78)

Address: 7156

```
property ain78_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN78_EF_READ_B (AIN#(0:149)_EF_READ_B@78)

Address: 7456

```
property ain78_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN78_EF_READ_C (AIN#(0:149)_EF_READ_C@78)

Address: 7756

```
property ain78_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN78_EF_READ_D (AIN#(0:149)_EF_READ_D@78)

Address: 8056

```
property ain79_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN79_BINARY (AIN#(0:254)_BINARY@79)

Address: 50158

```
property ain79_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@79)

Address: 9458

```
property ain79_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@79)

Address: 9758

```
property ain79_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@79)

Address: 10058

```
property ain79_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@79)

Address: 10358

```
property ain79_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@79)

Address: 10658

```
property ain79_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@79)

Address: 10958

```
property ain79_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@79)

Address: 11258

```
property ain79_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@79)

Address: 11558

```
property ain79_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@79)

Address: 11858

```
property ain79_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@79)

Address: 12158

```
property ain79_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN79_EF_INDEX (AIN#(0:149)_EF_INDEX@79)

Address: 9158

```
property ain79_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN79_EF_READ_A (AIN#(0:149)_EF_READ_A@79)

Address: 7158

```
property ain79_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN79_EF_READ_B (AIN#(0:149)_EF_READ_B@79)

Address: 7458

```
property ain79_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN79_EF_READ_C (AIN#(0:149)_EF_READ_C@79)

Address: 7758

```
property ain79_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN79_EF_READ_D (AIN#(0:149)_EF_READ_D@79)

Address: 8058

```
property ain7_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN7_BINARY (AIN#(0:254)_BINARY@7)

Address: 50014

```
property ain7_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@7)

Address: 9314

```
property ain7_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@7)

Address: 9614

```
property ain7_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@7)

Address: 9914

```
property ain7_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@7)

Address: 10214

```
property ain7_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@7)

Address: 10514

```
property ain7_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@7)

Address: 10814

```
property ain7_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@7)

Address: 11114

```
property ain7_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@7)

Address: 11414

```
property ain7_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@7)

Address: 11714

```
property ain7_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@7)

Address: 12014

```
property ain7_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN7_EF_INDEX (AIN#(0:149)_EF_INDEX@7)

Address: 9014

```
property ain7_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN7_EF_READ_A (AIN#(0:149)_EF_READ_A@7)

Address: 7014

```
property ain7_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN7_EF_READ_B (AIN#(0:149)_EF_READ_B@7)

Address: 7314

```
property ain7_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN7_EF_READ_C (AIN#(0:149)_EF_READ_C@7)

Address: 7614

```
property ain7_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN7_EF_READ_D (AIN#(0:149)_EF_READ_D@7)

Address: 7914

```
property ain8: float
```

Returns the voltage of the specified analog input.

Register: AIN8 (AIN#(0:11)@8)

Address: 16

```
property ain80_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN80_BINARY (AIN#(0:254)_BINARY@80)

Address: 50160

```
property ain80_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@80)

Address: 9460

```
property ain80_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@80)

Address: 9760

```
property ain80_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@80)

Address: 10060

```
property ain80_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@80)

Address: 10360

```
property ain80_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@80)

Address: 10660

```
property ain80_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@80)

Address: 10960

```
property ain80_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@80)

Address: 11260

```
property ain80_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@80)

Address: 11560

```
property ain80_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@80)

Address: 11860

```
property ain80_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@80)

Address: 12160

```
property ain80_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN80_EF_INDEX (AIN#(0:149)_EF_INDEX@80)

Address: 9160

```
property ain80_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN80_EF_READ_A (AIN#(0:149)_EF_READ_A@80)

Address: 7160

```
property ain80_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN80_EF_READ_B (AIN#(0:149)_EF_READ_B@80)

Address: 7460

```
property ain80_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN80_EF_READ_C (AIN#(0:149)_EF_READ_C@80)

Address: 7760

```
property ain80_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN80_EF_READ_D (AIN#(0:149)_EF_READ_D@80)

Address: 8060

```
property ain81_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN81_BINARY (AIN#(0:254)_BINARY@81)

Address: 50162

```
property ain81_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@81)

Address: 9462

```
property ain81_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@81)

Address: 9762

```
property ain81_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@81)

Address: 10062

```
property ain81_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@81)

Address: 10362

```
property ain81_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@81)

Address: 10662

```
property ain81_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@81)

Address: 10962

```
property ain81_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@81)

Address: 11262

```
property ain81_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@81)

Address: 11562

```
property ain81_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@81)

Address: 11862

```
property ain81_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@81)

Address: 12162

```
property ain81_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN81_EF_INDEX (AIN#(0:149)_EF_INDEX@81)

Address: 9162

```
property ain81_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN81_EF_READ_A (AIN#(0:149)_EF_READ_A@81)

Address: 7162

```
property ain81_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN81_EF_READ_B (AIN#(0:149)_EF_READ_B@81)

Address: 7462

```
property ain81_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN81_EF_READ_C (AIN#(0:149)_EF_READ_C@81)

Address: 7762

```
property ain81_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN81_EF_READ_D (AIN#(0:149)_EF_READ_D@81)

Address: 8062

```
property ain82_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN82_BINARY (AIN#(0:254)_BINARY@82)

Address: 50164

```
property ain82_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@82)

Address: 9464

```
property ain82_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@82)

Address: 9764

```
property ain82_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@82)

Address: 10064

```
property ain82_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@82)

Address: 10364

```
property ain82_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@82)

Address: 10664

```
property ain82_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@82)

Address: 10964

```
property ain82_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@82)

Address: 11264

```
property ain82_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@82)

Address: 11564

```
property ain82_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@82)

Address: 11864

```
property ain82_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@82)

Address: 12164

```
property ain82_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN82_EF_INDEX (AIN#(0:149)_EF_INDEX@82)

Address: 9164

```
property ain82_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN82_EF_READ_A (AIN#(0:149)_EF_READ_A@82)

Address: 7164

```
property ain82_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN82_EF_READ_B (AIN#(0:149)_EF_READ_B@82)

Address: 7464

```
property ain82_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN82_EF_READ_C (AIN#(0:149)_EF_READ_C@82)

Address: 7764

```
property ain82_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN82_EF_READ_D (AIN#(0:149)_EF_READ_D@82)

Address: 8064

```
property ain83_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN83_BINARY (AIN#(0:254)_BINARY@83)

Address: 50166

```
property ain83_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@83)

Address: 9466

```
property ain83_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@83)

Address: 9766

```
property ain83_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@83)

Address: 10066

```
property ain83_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@83)

Address: 10366

```
property ain83_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@83)

Address: 10666

```
property ain83_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@83)

Address: 10966

```
property ain83_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@83)

Address: 11266

```
property ain83_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@83)

Address: 11566

```
property ain83_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@83)

Address: 11866

```
property ain83_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@83)

Address: 12166

```
property ain83_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN83_EF_INDEX (AIN#(0:149)_EF_INDEX@83)

Address: 9166

```
property ain83_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN83_EF_READ_A (AIN#(0:149)_EF_READ_A@83)

Address: 7166

```
property ain83_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN83_EF_READ_B (AIN#(0:149)_EF_READ_B@83)

Address: 7466

```
property ain83_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN83_EF_READ_C (AIN#(0:149)_EF_READ_C@83)

Address: 7766

```
property ain83_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN83_EF_READ_D (AIN#(0:149)_EF_READ_D@83)

Address: 8066

```
property ain84_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN84_BINARY (AIN#(0:254)_BINARY@84)

Address: 50168

```
property ain84_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@84)

Address: 9468

```
property ain84_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@84)

Address: 9768

```
property ain84_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@84)

Address: 10068

```
property ain84_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@84)

Address: 10368

```
property ain84_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@84)

Address: 10668

```
property ain84_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@84)

Address: 10968

```
property ain84_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@84)

Address: 11268

```
property ain84_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@84)

Address: 11568

```
property ain84_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@84)

Address: 11868

```
property ain84_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@84)

Address: 12168

```
property ain84_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN84_EF_INDEX (AIN#(0:149)_EF_INDEX@84)

Address: 9168

```
property ain84_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN84_EF_READ_A (AIN#(0:149)_EF_READ_A@84)

Address: 7168

```
property ain84_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN84_EF_READ_B (AIN#(0:149)_EF_READ_B@84)

Address: 7468

```
property ain84_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN84_EF_READ_C (AIN#(0:149)_EF_READ_C@84)

Address: 7768

```
property ain84_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN84_EF_READ_D (AIN#(0:149)_EF_READ_D@84)

Address: 8068

```
property ain85_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN85_BINARY (AIN#(0:254)_BINARY@85)

Address: 50170

```
property ain85_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@85)

Address: 9470

```
property ain85_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@85)

Address: 9770

```
property ain85_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@85)

Address: 10070

```
property ain85_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@85)

Address: 10370

```
property ain85_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@85)

Address: 10670

```
property ain85_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@85)

Address: 10970

```
property ain85_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@85)

Address: 11270

```
property ain85_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@85)

Address: 11570

```
property ain85_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@85)

Address: 11870

```
property ain85_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@85)

Address: 12170

```
property ain85_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN85_EF_INDEX (AIN#(0:149)_EF_INDEX@85)

Address: 9170

```
property ain85_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN85_EF_READ_A (AIN#(0:149)_EF_READ_A@85)

Address: 7170

```
property ain85_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN85_EF_READ_B (AIN#(0:149)_EF_READ_B@85)

Address: 7470

```
property ain85_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN85_EF_READ_C (AIN#(0:149)_EF_READ_C@85)

Address: 7770

```
property ain85_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN85_EF_READ_D (AIN#(0:149)_EF_READ_D@85)

Address: 8070

```
property ain86_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN86_BINARY (AIN#(0:254)_BINARY@86)

Address: 50172

```
property ain86_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@86)

Address: 9472

```
property ain86_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@86)

Address: 9772

```
property ain86_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@86)

Address: 10072

```
property ain86_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@86)

Address: 10372

```
property ain86_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@86)

Address: 10672

```
property ain86_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@86)

Address: 10972

```
property ain86_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@86)

Address: 11272

```
property ain86_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@86)

Address: 11572

```
property ain86_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@86)

Address: 11872

```
property ain86_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@86)

Address: 12172

```
property ain86_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN86_EF_INDEX (AIN#(0:149)_EF_INDEX@86)

Address: 9172

```
property ain86_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN86_EF_READ_A (AIN#(0:149)_EF_READ_A@86)

Address: 7172

```
property ain86_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN86_EF_READ_B (AIN#(0:149)_EF_READ_B@86)

Address: 7472

```
property ain86_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN86_EF_READ_C (AIN#(0:149)_EF_READ_C@86)

Address: 7772

```
property ain86_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN86_EF_READ_D (AIN#(0:149)_EF_READ_D@86)

Address: 8072

```
property ain87_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN87_BINARY (AIN#(0:254)_BINARY@87)

Address: 50174

```
property ain87_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@87)

Address: 9474

```
property ain87_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@87)

Address: 9774

```
property ain87_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@87)

Address: 10074

```
property ain87_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@87)

Address: 10374

```
property ain87_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@87)

Address: 10674

```
property ain87_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@87)

Address: 10974

```
property ain87_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@87)

Address: 11274

```
property ain87_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@87)

Address: 11574

```
property ain87_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@87)

Address: 11874

```
property ain87_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@87)

Address: 12174

```
property ain87_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN87_EF_INDEX (AIN#(0:149)_EF_INDEX@87)

Address: 9174

```
property ain87_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN87_EF_READ_A (AIN#(0:149)_EF_READ_A@87)

Address: 7174

```
property ain87_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN87_EF_READ_B (AIN#(0:149)_EF_READ_B@87)

Address: 7474

```
property ain87_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN87_EF_READ_C (AIN#(0:149)_EF_READ_C@87)

Address: 7774

```
property ain87_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN87_EF_READ_D (AIN#(0:149)_EF_READ_D@87)

Address: 8074

```
property ain88_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN88_BINARY (AIN#(0:254)_BINARY@88)

Address: 50176

```
property ain88_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@88)

Address: 9476

```
property ain88_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@88)

Address: 9776

```
property ain88_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@88)

Address: 10076

```
property ain88_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@88)

Address: 10376

```
property ain88_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@88)

Address: 10676

```
property ain88_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@88)

Address: 10976

```
property ain88_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@88)

Address: 11276

```
property ain88_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@88)

Address: 11576

```
property ain88_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@88)

Address: 11876

```
property ain88_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@88)

Address: 12176

```
property ain88_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN88_EF_INDEX (AIN#(0:149)_EF_INDEX@88)

Address: 9176

```
property ain88_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN88_EF_READ_A (AIN#(0:149)_EF_READ_A@88)

Address: 7176

```
property ain88_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN88_EF_READ_B (AIN#(0:149)_EF_READ_B@88)

Address: 7476

```
property ain88_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN88_EF_READ_C (AIN#(0:149)_EF_READ_C@88)

Address: 7776

```
property ain88_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN88_EF_READ_D (AIN#(0:149)_EF_READ_D@88)

Address: 8076

```
property ain89_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN89_BINARY (AIN#(0:254)_BINARY@89)

Address: 50178

```
property ain89_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@89)

Address: 9478

```
property ain89_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@89)

Address: 9778

```
property ain89_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@89)

Address: 10078

```
property ain89_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@89)

Address: 10378

```
property ain89_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@89)

Address: 10678

```
property ain89_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@89)

Address: 10978

```
property ain89_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@89)

Address: 11278

```
property ain89_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@89)

Address: 11578

```
property ain89_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@89)

Address: 11878

```
property ain89_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@89)

Address: 12178

```
property ain89_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN89_EF_INDEX (AIN#(0:149)_EF_INDEX@89)

Address: 9178

```
property ain89_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN89_EF_READ_A (AIN#(0:149)_EF_READ_A@89)

Address: 7178

```
property ain89_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN89_EF_READ_B (AIN#(0:149)_EF_READ_B@89)

Address: 7478

```
property ain89_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN89_EF_READ_C (AIN#(0:149)_EF_READ_C@89)

Address: 7778

```
property ain89_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN89_EF_READ_D (AIN#(0:149)_EF_READ_D@89)

Address: 8078

```
property ain8_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN8_BINARY (AIN#(0:254)_BINARY@8)

Address: 50016

```
property ain8_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@8)

Address: 9316

```
property ain8_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@8)

Address: 9616

```
property ain8_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@8)

Address: 9916

```
property ain8_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@8)

Address: 10216

```
property ain8_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@8)

Address: 10516

```
property ain8_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@8)

Address: 10816

```
property ain8_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@8)

Address: 11116

```
property ain8_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@8)

Address: 11416

```
property ain8_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@8)

Address: 11716

```
property ain8_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@8)

Address: 12016

```
property ain8_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN8_EF_INDEX (AIN#(0:149)_EF_INDEX@8)

Address: 9016

```
property ain8_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN8_EF_READ_A (AIN#(0:149)_EF_READ_A@8)

Address: 7016

```
property ain8_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN8_EF_READ_B (AIN#(0:149)_EF_READ_B@8)

Address: 7316

```
property ain8_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN8_EF_READ_C (AIN#(0:149)_EF_READ_C@8)

Address: 7616

```
property ain8_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN8_EF_READ_D (AIN#(0:149)_EF_READ_D@8)

Address: 7916

```
property ain9: float
```

Returns the voltage of the specified analog input.

Register: AIN9 (AIN#(0:11)@9)

Address: 18

```
property ain90_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN90_BINARY (AIN#(0:254)_BINARY@90)

Address: 50180

```
property ain90_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@90)

Address: 9480

```
property ain90_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@90)

Address: 9780

```
property ain90_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@90)

Address: 10080

```
property ain90_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@90)

Address: 10380

```
property ain90_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@90)

Address: 10680

```
property ain90_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@90)

Address: 10980

```
property ain90_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@90)

Address: 11280

```
property ain90_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@90)

Address: 11580

```
property ain90_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@90)

Address: 11880

```
property ain90_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@90)

Address: 12180

```
property ain90_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN90_EF_INDEX (AIN#(0:149)_EF_INDEX@90)

Address: 9180

```
property ain90_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN90_EF_READ_A (AIN#(0:149)_EF_READ_A@90)

Address: 7180

```
property ain90_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN90_EF_READ_B (AIN#(0:149)_EF_READ_B@90)

Address: 7480

```
property ain90_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN90_EF_READ_C (AIN#(0:149)_EF_READ_C@90)

Address: 7780

```
property ain90_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN90_EF_READ_D (AIN#(0:149)_EF_READ_D@90)

Address: 8080

```
property ain91_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN91_BINARY (AIN#(0:254)_BINARY@91)

Address: 50182

```
property ain91_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@91)

Address: 9482

```
property ain91_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@91)

Address: 9782

```
property ain91_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@91)

Address: 10082

```
property ain91_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@91)

Address: 10382

```
property ain91_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@91)

Address: 10682

```
property ain91_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@91)

Address: 10982

```
property ain91_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@91)

Address: 11282

```
property ain91_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@91)

Address: 11582

```
property ain91_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@91)

Address: 11882

```
property ain91_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@91)

Address: 12182

```
property ain91_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN91_EF_INDEX (AIN#(0:149)_EF_INDEX@91)

Address: 9182

```
property ain91_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN91_EF_READ_A (AIN#(0:149)_EF_READ_A@91)

Address: 7182

```
property ain91_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN91_EF_READ_B (AIN#(0:149)_EF_READ_B@91)

Address: 7482

```
property ain91_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN91_EF_READ_C (AIN#(0:149)_EF_READ_C@91)

Address: 7782

```
property ain91_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN91_EF_READ_D (AIN#(0:149)_EF_READ_D@91)

Address: 8082

```
property ain92_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN92_BINARY (AIN#(0:254)_BINARY@92)

Address: 50184

```
property ain92_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@92)

Address: 9484

```
property ain92_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@92)

Address: 9784

```
property ain92_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@92)

Address: 10084

```
property ain92_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@92)

Address: 10384

```
property ain92_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@92)

Address: 10684

```
property ain92_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@92)

Address: 10984

```
property ain92_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@92)

Address: 11284

```
property ain92_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@92)

Address: 11584

```
property ain92_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@92)

Address: 11884

```
property ain92_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@92)

Address: 12184

```
property ain92_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN92_EF_INDEX (AIN#(0:149)_EF_INDEX@92)

Address: 9184

```
property ain92_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN92_EF_READ_A (AIN#(0:149)_EF_READ_A@92)

Address: 7184

```
property ain92_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN92_EF_READ_B (AIN#(0:149)_EF_READ_B@92)

Address: 7484

```
property ain92_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN92_EF_READ_C (AIN#(0:149)_EF_READ_C@92)

Address: 7784

```
property ain92_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN92_EF_READ_D (AIN#(0:149)_EF_READ_D@92)

Address: 8084

```
property ain93_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN93_BINARY (AIN#(0:254)_BINARY@93)

Address: 50186

```
property ain93_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@93)

Address: 9486

```
property ain93_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@93)

Address: 9786

```
property ain93_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@93)

Address: 10086

```
property ain93_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@93)

Address: 10386

```
property ain93_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@93)

Address: 10686

```
property ain93_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@93)

Address: 10986

```
property ain93_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@93)

Address: 11286

```
property ain93_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@93)

Address: 11586

```
property ain93_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@93)

Address: 11886

```
property ain93_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@93)

Address: 12186

```
property ain93_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN93_EF_INDEX (AIN#(0:149)_EF_INDEX@93)

Address: 9186

```
property ain93_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN93_EF_READ_A (AIN#(0:149)_EF_READ_A@93)

Address: 7186

```
property ain93_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN93_EF_READ_B (AIN#(0:149)_EF_READ_B@93)

Address: 7486

```
property ain93_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN93_EF_READ_C (AIN#(0:149)_EF_READ_C@93)

Address: 7786

```
property ain93_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN93_EF_READ_D (AIN#(0:149)_EF_READ_D@93)

Address: 8086

```
property ain94_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN94_BINARY (AIN#(0:254)_BINARY@94)

Address: 50188

```
property ain94_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@94)

Address: 9488

```
property ain94_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@94)

Address: 9788

```
property ain94_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@94)

Address: 10088

```
property ain94_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@94)

Address: 10388

```
property ain94_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@94)

Address: 10688

```
property ain94_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@94)

Address: 10988

```
property ain94_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@94)

Address: 11288

```
property ain94_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@94)

Address: 11588

```
property ain94_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@94)

Address: 11888

```
property ain94_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@94)

Address: 12188

```
property ain94_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN94_EF_INDEX (AIN#(0:149)_EF_INDEX@94)

Address: 9188

```
property ain94_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN94_EF_READ_A (AIN#(0:149)_EF_READ_A@94)

Address: 7188

```
property ain94_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN94_EF_READ_B (AIN#(0:149)_EF_READ_B@94)

Address: 7488

```
property ain94_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN94_EF_READ_C (AIN#(0:149)_EF_READ_C@94)

Address: 7788

```
property ain94_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN94_EF_READ_D (AIN#(0:149)_EF_READ_D@94)

Address: 8088

```
property ain95_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN95_BINARY (AIN#(0:254)_BINARY@95)

Address: 50190

```
property ain95_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@95)

Address: 9490

```
property ain95_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@95)

Address: 9790

```
property ain95_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@95)

Address: 10090

```
property ain95_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@95)

Address: 10390

```
property ain95_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@95)

Address: 10690

```
property ain95_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@95)

Address: 10990

```
property ain95_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@95)

Address: 11290

```
property ain95_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@95)

Address: 11590

```
property ain95_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@95)

Address: 11890

```
property ain95_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@95)

Address: 12190

```
property ain95_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN95_EF_INDEX (AIN#(0:149)_EF_INDEX@95)

Address: 9190

```
property ain95_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN95_EF_READ_A (AIN#(0:149)_EF_READ_A@95)

Address: 7190

```
property ain95_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN95_EF_READ_B (AIN#(0:149)_EF_READ_B@95)

Address: 7490

```
property ain95_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN95_EF_READ_C (AIN#(0:149)_EF_READ_C@95)

Address: 7790

```
property ain95_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN95_EF_READ_D (AIN#(0:149)_EF_READ_D@95)

Address: 8090

```
property ain96_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN96_BINARY (AIN#(0:254)_BINARY@96)

Address: 50192

```
property ain96_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@96)

Address: 9492

```
property ain96_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@96)

Address: 9792

```
property ain96_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@96)

Address: 10092

```
property ain96_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@96)

Address: 10392

```
property ain96_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@96)

Address: 10692

```
property ain96_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@96)

Address: 10992

```
property ain96_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@96)

Address: 11292

```
property ain96_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@96)

Address: 11592

```
property ain96_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@96)

Address: 11892

```
property ain96_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@96)

Address: 12192

```
property ain96_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN96_EF_INDEX (AIN#(0:149)_EF_INDEX@96)

Address: 9192

```
property ain96_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN96_EF_READ_A (AIN#(0:149)_EF_READ_A@96)

Address: 7192

```
property ain96_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN96_EF_READ_B (AIN#(0:149)_EF_READ_B@96)

Address: 7492

```
property ain96_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN96_EF_READ_C (AIN#(0:149)_EF_READ_C@96)

Address: 7792

```
property ain96_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN96_EF_READ_D (AIN#(0:149)_EF_READ_D@96)

Address: 8092

```
property ain97_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN97_BINARY (AIN#(0:254)_BINARY@97)

Address: 50194

```
property ain97_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@97)

Address: 9494

```
property ain97_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@97)

Address: 9794

```
property ain97_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@97)

Address: 10094

```
property ain97_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@97)

Address: 10394

```
property ain97_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@97)

Address: 10694

```
property ain97_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@97)

Address: 10994

```
property ain97_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@97)

Address: 11294

```
property ain97_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@97)

Address: 11594

```
property ain97_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@97)

Address: 11894

```
property ain97_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@97)

Address: 12194

```
property ain97_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN97_EF_INDEX (AIN#(0:149)_EF_INDEX@97)

Address: 9194

```
property ain97_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN97_EF_READ_A (AIN#(0:149)_EF_READ_A@97)

Address: 7194

```
property ain97_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN97_EF_READ_B (AIN#(0:149)_EF_READ_B@97)

Address: 7494

```
property ain97_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN97_EF_READ_C (AIN#(0:149)_EF_READ_C@97)

Address: 7794

```
property ain97_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN97_EF_READ_D (AIN#(0:149)_EF_READ_D@97)

Address: 8094

```
property ain98_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN98_BINARY (AIN#(0:254)_BINARY@98)

Address: 50196

```
property ain98_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@98)

Address: 9496

```
property ain98_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@98)

Address: 9796

```
property ain98_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@98)

Address: 10096

```
property ain98_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@98)

Address: 10396

```
property ain98_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@98)

Address: 10696

```
property ain98_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@98)

Address: 10996

```
property ain98_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@98)

Address: 11296

```
property ain98_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@98)

Address: 11596

```
property ain98_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@98)

Address: 11896

```
property ain98_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@98)

Address: 12196

```
property ain98_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN98_EF_INDEX (AIN#(0:149)_EF_INDEX@98)

Address: 9196

```
property ain98_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN98_EF_READ_A (AIN#(0:149)_EF_READ_A@98)

Address: 7196

```
property ain98_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN98_EF_READ_B (AIN#(0:149)_EF_READ_B@98)

Address: 7496

```
property ain98_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN98_EF_READ_C (AIN#(0:149)_EF_READ_C@98)

Address: 7796

```
property ain98_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN98_EF_READ_D (AIN#(0:149)_EF_READ_D@98)

Address: 8096

```
property ain99_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN99_BINARY (AIN#(0:254)_BINARY@99)

Address: 50198

```
property ain99_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@99)

Address: 9498

```
property ain99_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@99)

Address: 9798

```
property ain99_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@99)

Address: 10098

```
property ain99_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@99)

Address: 10398

```
property ain99_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@99)

Address: 10698

```
property ain99_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@99)

Address: 10998

```
property ain99_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@99)

Address: 11298

```
property ain99_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@99)

Address: 11598

```
property ain99_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@99)

Address: 11898

```
property ain99_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@99)

Address: 12198

```
property ain99_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN99_EF_INDEX (AIN#(0:149)_EF_INDEX@99)

Address: 9198

```
property ain99_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN99_EF_READ_A (AIN#(0:149)_EF_READ_A@99)

Address: 7198

```
property ain99_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN99_EF_READ_B (AIN#(0:149)_EF_READ_B@99)

Address: 7498

```
property ain99_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN99_EF_READ_C (AIN#(0:149)_EF_READ_C@99)

Address: 7798

```
property ain99_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN99_EF_READ_D (AIN#(0:149)_EF_READ_D@99)

Address: 8098

```
property ain9_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN9_BINARY (AIN#(0:254)_BINARY@9)

Address: 50018

```
property ain9_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@9)

Address: 9318

```
property ain9_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@9)

Address: 9618

```
property ain9_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@9)

Address: 9918

```
property ain9_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@9)

Address: 10218

```
property ain9_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@9)

Address: 10518

```
property ain9_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@9)

Address: 10818

```
property ain9_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@9)

Address: 11118

```
property ain9_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@9)

Address: 11418

```
property ain9_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@9)

Address: 11718

```
property ain9_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@9)

Address: 12018

```
property ain9_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN9_EF_INDEX (AIN#(0:149)_EF_INDEX@9)

Address: 9018

```
property ain9_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN9_EF_READ_A (AIN#(0:149)_EF_READ_A@9)

Address: 7018

```
property ain9_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN9_EF_READ_B (AIN#(0:149)_EF_READ_B@9)

Address: 7318

```
property ain9_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN9_EF_READ_C (AIN#(0:149)_EF_READ_C@9)

Address: 7618

```
property ain9_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN9_EF_READ_D (AIN#(0:149)_EF_READ_D@9)

Address: 7918

```
property ain_all_ef_index: int
```

Write 0 to deactivate AIN_EF on all AINs. No other values may be written to this register. Reads will return the AIN_EF index if all 128 AINs are set to the same value. If values are not the same returns 0xFFFF (65535).

Register: AIN_ALL_EF_INDEX

Address: 43906

```
property ain_all_negative_ch: int
```

A write to this global parameter affects all AIN. Writing 1 will set all AINs to differential. Writing 199 will set all AINs to single-ended. A read will return 1 if all AINs are set to differential and 199 if all AINs are set to single-ended. If AIN configurations are not consistent 0xFFFF will be returned.

Register: AIN_ALL_NEGATIVE_CH

Address: 43902

```
property ain_all_range: float
```

A write to this global parameter affects all AIN. A read will return the correct setting if all channels are set the same, but otherwise will return -9999.

Register: AIN_ALL_RANGE

Address: 43900

```
property ain_all_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times. A write to this global parameter affects all AIN. A read will return the correct setting if all channels are set the same, but otherwise will return 0xFFFF.

Register: AIN_ALL_RESOLUTION_INDEX

Address: 43903

```
property ain_all_settling_us: float
```

Settling time for command-response and AIN-EF readings. A write to this global parameter affects all AIN. A read will return the correct setting if all channels are set the same, but otherwise will return -9999. Max is 50,000 us.

Register: AIN_ALL_SETTLING_US

Address: 43904

```
property asynch_baud: int
```

The symbol rate that will be used for communication. 9600 is typical. Up to 38400 works, but heavily loads the T7's processor.

Register: ASYNCH_BAUD

Address: 5420

```
property asynch_data_rx: int
```

Read received data from here.

This register is a buffer. Underrun behavior - fill with zeros.

Register: ASYNCH_DATA_RX

Address: 5495

```
property asynch_data_tx: int
```

Write data to be transmitted here. This register is a buffer.

Register: ASYNCH_DATA_TX

Address: 5490

```
property asynch_enable: int
```

1 = Turn on Asynch. Configures timing hardware, DIO lines and allocates the receiving buffer.

Register: ASYNCH_ENABLE

Address: 5400

```
property asynch_num_bytes_rx: int
```

The number of data bytes that have been received.

Register: ASYNCH_NUM_BYTES_RX

Address: 5435

```
property asynch_num_bytes_tx: int
```

The number of bytes to be transmitted after writing to GO. Max is 256.

Register: ASYNCH_NUM_BYTES_TX

Address: 5440

```
property asynch_num_data_bits: int
```

The number of data bits per frame. 0-8, 0=8.

Register: ASYNCH_NUM_DATA_BITS

Address: 5415

```
property asynch_num_parity_errors: int
```

The number of parity errors that have been detected. Cleared when UART is enabled. Can also be cleared by writing 0.

Register: ASYNCH_NUM_PARITY_ERRORS

Address: 5465

```
property asynch_num_stop_bits: int
```

The number of stop bits. Values: 0 = zero stop bits, 1 = one stop bit, 2 = two stop bits.

Register: ASYNCH_NUM_STOP_BITS

Address: 5455

```
property asynch_parity: int
```

Parity setting: 0=none, 1=odd, 2=even.

Register: ASYNCH_PARITY

Address: 5460

```
property asynch_rx_buffer_size_bytes: int
```

Number of bytes to use for the receiving buffer. Max is 2048. 0 = 200.

Register: ASYNCH_RX_BUFFER_SIZE_BYTES

Address: 5430

```
property asynch_rx_dionum: int
```

The DIO line that will receive data. (RX)

Register: ASYNCH_RX_DIONUM

Address: 5405

```
property asynch_tx_dionum: int
```

The DIO line that will transmit data. (TX)

Register: ASYNCH_TX_DIONUM

Address: 5410

```
property asynch_tx_go: int
```

Write a 1 to this register to initiate a transmission.

Register: ASYNCH_TX_GO

Address: 5450

```
property bootloader_version: float
```

The bootloader version installed on the main processor.

Register: BOOTLOADER_VERSION

Address: 60006

```
property cio0: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: CIO0 (CIO#(0:3)@0)

Address: 2016

```
property cio1: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: CIO1 (CIO#(0:3)@1)

Address: 2017

```
property cio2: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: CIO2 (CIO#(0:3)@2)

Address: 2018

```
property cio3: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: CIO3 (CIO#(0:3)@3)

Address: 2019

```
property cio_direction: int
```

Read or write the direction of the 4 bits of CIO in a single binary-encoded value. 0=Input and 1=Output. The upper 8-bits of this value are inhibits.

Register: CIO_DIRECTION

Address: 2602

```
property cio_mio_state: int
```

Read or write the state of the 12 bits of CIO-MIO in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. Reading lines set to output returns the current logic levels on the terminals, not necessarily the output states written.

Register: CIO_MIO_STATE

Address: 2582

```
property cio_state: int
```

Read or write the state of the 4 bits of CIO in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. Reading lines set to output returns the current logic levels on the terminals, not necessarily the output states written. The upper 8-bits of this value are inhibits.

Register: CIO_STATE

Address: 2502

```
property cleanse: int
```

Writing 0x5317052E to this register will trigger a system cleanse.

Register: CLEANSE

Address: 49090

```
property core_timer: int
```

Internal 32-bit system timer running at 1/2 core speed, thus normally 80M/2 => 40 MHz.

Register: CORE_TIMER

Address: 61520

```
property dac0: float
```

Pass a voltage for the specified analog output.

Register: DAC0 (DAC#(0:1)@0)

Address: 1000

```
property dac0_binary: int
```

Writes binary values to the DACs. Binary values are 16-bit. If the DAC's resolution is less than 16 then the lower bits are ignored. 0 = lowest output, 65535 = highest output.

Register: DAC0_BINARY (DAC#(0:1)_BINARY@0)

Address: 51000

```
property dac1: float
```

Pass a voltage for the specified analog output.

Register: DAC1 (DAC#(0:1)@1)

Address: 1002

```
property dac1_binary: int
```

Writes binary values to the DACs. Binary values are 16-bit. If the DAC's resolution is less than 16 then the lower bits are ignored. 0 = lowest output, 65535 = highest output.

Register: DAC1_BINARY (DAC#(0:1)_BINARY@1)

Address: 51002

```
property dac1_frequency_out_enable: int
```

0 = off, 1 = output 10 Hz signal on DAC1. The signal will be a square wave with peaks of 0 and 3.3V. Note that writing to DAC1 or enabling a stream out which is targeting DAC1 will disable this feature.

Register: DAC1_FREQUENCY_OUT_ENABLE

Address: 61532

```
property device_name_default: str
```

Reads return the current device name. Writes update the default and current device name. A reboot is necessary to update the name reported by NBNS. Up to 49 characters, cannot contain periods.

Register: DEVICE_NAME_DEFAULT

Address: 60500

```
property device_reset_dbg_reg: int
```

Bitmask register that indicates why a device was last re-set. bit0: Power-on reset, bit1: Brown-out, bit2: Wake from idle, bit3: Wake from sleep, bit4: watchdog timer time-out, bit5: reserved, bit6: Software reset, bit7: external reset (MCLR), bit8: Voltage regulator standby, bit9: Configuration mismatch.

Register: DEVICE_RESET_DBG_REG

Address: 60014

```
property dio0_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO0_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@0)

Address: 44300

```
property dio0_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO0_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@0)

Address: 44400

```
property dio0_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO0_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@0)

Address: 44500

```
property dio0_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO0_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@0)

Address: 44600

```
property dio0_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO0_EF_ENABLE (DIO#(0:22)_EF_ENABLE@0)

Address: 44000

```
property dio0_ef_index: int
```

An index to specify the feature you want.

Register: DIO0_EF_INDEX (DIO#(0:22)_EF_INDEX@0)

Address: 44100

```
property dio0_ef_options: int
```

Function dependent on selected feature index.

Register: DIO0_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@0)

Address: 44200

```
property dio0_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO0_EF_READ_A (DIO#(0:22)_EF_READ_A@0)

Address: 3000

```
property dio0_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO0_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@0)

Address: 3100

Type:
Reads the same value as DIO#(0

```
property dio0_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO0_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@0)

Address: 3500

```
property dio0_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO0_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@0)

Address: 3600

```
property dio0_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO0_EF_READ_B (DIO#(0:22)_EF_READ_B@0)

Address: 3200

```
property dio0_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO0_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@0)

Address: 3700

```
property dio10_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO10_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@10)

Address: 44320

```
property dio10_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO10_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@10)

Address: 44420

```
property dio10_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO10_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@10)

Address: 44520

```
property dio10_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO10_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@10)

Address: 44620

```
property dio10_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO10_EF_ENABLE (DIO#(0:22)_EF_ENABLE@10)

Address: 44020

```
property dio10_ef_index: int
```

An index to specify the feature you want.

Register: DIO10_EF_INDEX (DIO#(0:22)_EF_INDEX@10)

Address: 44120

```
property dio10_ef_options: int
```

Function dependent on selected feature index.

Register: DIO10_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@10)

Address: 44220

```
property dio10_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO10_EF_READ_A (DIO#(0:22)_EF_READ_A@10)

Address: 3020

```
property dio10_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO10_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@10)

Address: 3120

Type:
Reads the same value as DIO#(0

```
property dio10_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO10_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@10)

Address: 3520

```
property dio10_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO10_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@10)

Address: 3620

```
property dio10_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO10_EF_READ_B (DIO#(0:22)_EF_READ_B@10)

Address: 3220

```
property dio10_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO10_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@10)

Address: 3720

```
property dio11_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO11_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@11)

Address: 44322

```
property dio11_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO11_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@11)

Address: 44422

```
property dio11_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO11_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@11)

Address: 44522

```
property dio11_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO11_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@11)

Address: 44622

```
property dio11_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO11_EF_ENABLE (DIO#(0:22)_EF_ENABLE@11)

Address: 44022

```
property dio11_ef_index: int
```

An index to specify the feature you want.

Register: DIO11_EF_INDEX (DIO#(0:22)_EF_INDEX@11)

Address: 44122

```
property dio11_ef_options: int
```

Function dependent on selected feature index.

Register: DIO11_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@11)

Address: 44222

```
property dio11_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO11_EF_READ_A (DIO#(0:22)_EF_READ_A@11)

Address: 3022

```
property dio11_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO11_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@11)

Address: 3122

Type:
Reads the same value as DIO#(0

```
property dio11_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO11_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@11)

Address: 3522

```
property dio11_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO11_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@11)

Address: 3622

```
property dio11_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO11_EF_READ_B (DIO#(0:22)_EF_READ_B@11)

Address: 3222

```
property dio11_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO11_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@11)

Address: 3722

```
property dio12_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO12_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@12)

Address: 44324

```
property dio12_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO12_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@12)

Address: 44424

```
property dio12_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO12_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@12)

Address: 44524

```
property dio12_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO12_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@12)

Address: 44624

```
property dio12_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO12_EF_ENABLE (DIO#(0:22)_EF_ENABLE@12)

Address: 44024

```
property dio12_ef_index: int
```

An index to specify the feature you want.

Register: DIO12_EF_INDEX (DIO#(0:22)_EF_INDEX@12)

Address: 44124

```
property dio12_ef_options: int
```

Function dependent on selected feature index.

Register: DIO12_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@12)

Address: 44224

```
property dio12_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO12_EF_READ_A (DIO#(0:22)_EF_READ_A@12)

Address: 3024

```
property dio12_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO12_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@12)

Address: 3124

Type:
Reads the same value as DIO#(0

```
property dio12_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO12_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@12)

Address: 3524

```
property dio12_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO12_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@12)

Address: 3624

```
property dio12_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO12_EF_READ_B (DIO#(0:22)_EF_READ_B@12)

Address: 3224

```
property dio12_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO12_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@12)

Address: 3724

```
property dio13_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO13_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@13)

Address: 44326

```
property dio13_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO13_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@13)

Address: 44426

```
property dio13_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO13_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@13)

Address: 44526

```
property dio13_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO13_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@13)

Address: 44626

```
property dio13_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO13_EF_ENABLE (DIO#(0:22)_EF_ENABLE@13)

Address: 44026

```
property dio13_ef_index: int
```

An index to specify the feature you want.

Register: DIO13_EF_INDEX (DIO#(0:22)_EF_INDEX@13)

Address: 44126

```
property dio13_ef_options: int
```

Function dependent on selected feature index.

Register: DIO13_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@13)

Address: 44226

```
property dio13_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO13_EF_READ_A (DIO#(0:22)_EF_READ_A@13)

Address: 3026

```
property dio13_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO13_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@13)

Address: 3126

Type:
Reads the same value as DIO#(0

```
property dio13_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO13_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@13)

Address: 3526

```
property dio13_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO13_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@13)

Address: 3626

```
property dio13_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO13_EF_READ_B (DIO#(0:22)_EF_READ_B@13)

Address: 3226

```
property dio13_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO13_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@13)

Address: 3726

```
property dio14_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO14_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@14)

Address: 44328

```
property dio14_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO14_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@14)

Address: 44428

```
property dio14_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO14_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@14)

Address: 44528

```
property dio14_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO14_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@14)

Address: 44628

```
property dio14_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO14_EF_ENABLE (DIO#(0:22)_EF_ENABLE@14)

Address: 44028

```
property dio14_ef_index: int
```

An index to specify the feature you want.

Register: DIO14_EF_INDEX (DIO#(0:22)_EF_INDEX@14)

Address: 44128

```
property dio14_ef_options: int
```

Function dependent on selected feature index.

Register: DIO14_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@14)

Address: 44228

```
property dio14_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO14_EF_READ_A (DIO#(0:22)_EF_READ_A@14)

Address: 3028

```
property dio14_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO14_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@14)

Address: 3128

Type:
Reads the same value as DIO#(0

```
property dio14_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO14_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@14)

Address: 3528

```
property dio14_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO14_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@14)

Address: 3628

```
property dio14_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO14_EF_READ_B (DIO#(0:22)_EF_READ_B@14)

Address: 3228

```
property dio14_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO14_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@14)

Address: 3728

```
property dio15_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO15_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@15)

Address: 44330

```
property dio15_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO15_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@15)

Address: 44430

```
property dio15_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO15_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@15)

Address: 44530

```
property dio15_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO15_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@15)

Address: 44630

```
property dio15_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO15_EF_ENABLE (DIO#(0:22)_EF_ENABLE@15)

Address: 44030

```
property dio15_ef_index: int
```

An index to specify the feature you want.

Register: DIO15_EF_INDEX (DIO#(0:22)_EF_INDEX@15)

Address: 44130

```
property dio15_ef_options: int
```

Function dependent on selected feature index.

Register: DIO15_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@15)

Address: 44230

```
property dio15_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO15_EF_READ_A (DIO#(0:22)_EF_READ_A@15)

Address: 3030

```
property dio15_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO15_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@15)

Address: 3130

Type:
Reads the same value as DIO#(0

```
property dio15_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO15_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@15)

Address: 3530

```
property dio15_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO15_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@15)

Address: 3630

```
property dio15_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO15_EF_READ_B (DIO#(0:22)_EF_READ_B@15)

Address: 3230

```
property dio15_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO15_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@15)

Address: 3730

```
property dio16_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO16_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@16)

Address: 44332

```
property dio16_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO16_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@16)

Address: 44432

```
property dio16_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO16_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@16)

Address: 44532

```
property dio16_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO16_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@16)

Address: 44632

```
property dio16_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO16_EF_ENABLE (DIO#(0:22)_EF_ENABLE@16)

Address: 44032

```
property dio16_ef_index: int
```

An index to specify the feature you want.

Register: DIO16_EF_INDEX (DIO#(0:22)_EF_INDEX@16)

Address: 44132

```
property dio16_ef_options: int
```

Function dependent on selected feature index.

Register: DIO16_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@16)

Address: 44232

```
property dio16_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO16_EF_READ_A (DIO#(0:22)_EF_READ_A@16)

Address: 3032

```
property dio16_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO16_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@16)

Address: 3132

Type:
Reads the same value as DIO#(0

```
property dio16_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO16_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@16)

Address: 3532

```
property dio16_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO16_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@16)

Address: 3632

```
property dio16_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO16_EF_READ_B (DIO#(0:22)_EF_READ_B@16)

Address: 3232

```
property dio16_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO16_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@16)

Address: 3732

```
property dio17_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO17_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@17)

Address: 44334

```
property dio17_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO17_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@17)

Address: 44434

```
property dio17_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO17_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@17)

Address: 44534

```
property dio17_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO17_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@17)

Address: 44634

```
property dio17_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO17_EF_ENABLE (DIO#(0:22)_EF_ENABLE@17)

Address: 44034

```
property dio17_ef_index: int
```

An index to specify the feature you want.

Register: DIO17_EF_INDEX (DIO#(0:22)_EF_INDEX@17)

Address: 44134

```
property dio17_ef_options: int
```

Function dependent on selected feature index.

Register: DIO17_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@17)

Address: 44234

```
property dio17_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO17_EF_READ_A (DIO#(0:22)_EF_READ_A@17)

Address: 3034

```
property dio17_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO17_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@17)

Address: 3134

Type:
Reads the same value as DIO#(0

```
property dio17_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO17_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@17)

Address: 3534

```
property dio17_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO17_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@17)

Address: 3634

```
property dio17_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO17_EF_READ_B (DIO#(0:22)_EF_READ_B@17)

Address: 3234

```
property dio17_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO17_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@17)

Address: 3734

```
property dio18_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO18_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@18)

Address: 44336

```
property dio18_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO18_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@18)

Address: 44436

```
property dio18_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO18_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@18)

Address: 44536

```
property dio18_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO18_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@18)

Address: 44636

```
property dio18_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO18_EF_ENABLE (DIO#(0:22)_EF_ENABLE@18)

Address: 44036

```
property dio18_ef_index: int
```

An index to specify the feature you want.

Register: DIO18_EF_INDEX (DIO#(0:22)_EF_INDEX@18)

Address: 44136

```
property dio18_ef_options: int
```

Function dependent on selected feature index.

Register: DIO18_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@18)

Address: 44236

```
property dio18_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO18_EF_READ_A (DIO#(0:22)_EF_READ_A@18)

Address: 3036

```
property dio18_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO18_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@18)

Address: 3136

Type:
Reads the same value as DIO#(0

```
property dio18_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO18_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@18)

Address: 3536

```
property dio18_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO18_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@18)

Address: 3636

```
property dio18_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO18_EF_READ_B (DIO#(0:22)_EF_READ_B@18)

Address: 3236

```
property dio18_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO18_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@18)

Address: 3736

```
property dio19_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO19_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@19)

Address: 44338

```
property dio19_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO19_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@19)

Address: 44438

```
property dio19_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO19_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@19)

Address: 44538

```
property dio19_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO19_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@19)

Address: 44638

```
property dio19_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO19_EF_ENABLE (DIO#(0:22)_EF_ENABLE@19)

Address: 44038

```
property dio19_ef_index: int
```

An index to specify the feature you want.

Register: DIO19_EF_INDEX (DIO#(0:22)_EF_INDEX@19)

Address: 44138

```
property dio19_ef_options: int
```

Function dependent on selected feature index.

Register: DIO19_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@19)

Address: 44238

```
property dio19_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO19_EF_READ_A (DIO#(0:22)_EF_READ_A@19)

Address: 3038

```
property dio19_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO19_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@19)

Address: 3138

Type:
Reads the same value as DIO#(0

```
property dio19_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO19_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@19)

Address: 3538

```
property dio19_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO19_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@19)

Address: 3638

```
property dio19_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO19_EF_READ_B (DIO#(0:22)_EF_READ_B@19)

Address: 3238

```
property dio19_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO19_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@19)

Address: 3738

```
property dio1_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO1_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@1)

Address: 44302

```
property dio1_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO1_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@1)

Address: 44402

```
property dio1_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO1_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@1)

Address: 44502

```
property dio1_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO1_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@1)

Address: 44602

```
property dio1_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO1_EF_ENABLE (DIO#(0:22)_EF_ENABLE@1)

Address: 44002

```
property dio1_ef_index: int
```

An index to specify the feature you want.

Register: DIO1_EF_INDEX (DIO#(0:22)_EF_INDEX@1)

Address: 44102

```
property dio1_ef_options: int
```

Function dependent on selected feature index.

Register: DIO1_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@1)

Address: 44202

```
property dio1_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO1_EF_READ_A (DIO#(0:22)_EF_READ_A@1)

Address: 3002

```
property dio1_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO1_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@1)

Address: 3102

Type:
Reads the same value as DIO#(0

```
property dio1_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO1_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@1)

Address: 3502

```
property dio1_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO1_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@1)

Address: 3602

```
property dio1_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO1_EF_READ_B (DIO#(0:22)_EF_READ_B@1)

Address: 3202

```
property dio1_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO1_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@1)

Address: 3702

```
property dio20_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO20_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@20)

Address: 44340

```
property dio20_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO20_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@20)

Address: 44440

```
property dio20_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO20_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@20)

Address: 44540

```
property dio20_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO20_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@20)

Address: 44640

```
property dio20_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO20_EF_ENABLE (DIO#(0:22)_EF_ENABLE@20)

Address: 44040

```
property dio20_ef_index: int
```

An index to specify the feature you want.

Register: DIO20_EF_INDEX (DIO#(0:22)_EF_INDEX@20)

Address: 44140

```
property dio20_ef_options: int
```

Function dependent on selected feature index.

Register: DIO20_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@20)

Address: 44240

```
property dio20_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO20_EF_READ_A (DIO#(0:22)_EF_READ_A@20)

Address: 3040

```
property dio20_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO20_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@20)

Address: 3140

Type:
Reads the same value as DIO#(0

```
property dio20_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO20_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@20)

Address: 3540

```
property dio20_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO20_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@20)

Address: 3640

```
property dio20_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO20_EF_READ_B (DIO#(0:22)_EF_READ_B@20)

Address: 3240

```
property dio20_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO20_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@20)

Address: 3740

```
property dio21_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO21_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@21)

Address: 44342

```
property dio21_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO21_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@21)

Address: 44442

```
property dio21_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO21_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@21)

Address: 44542

```
property dio21_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO21_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@21)

Address: 44642

```
property dio21_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO21_EF_ENABLE (DIO#(0:22)_EF_ENABLE@21)

Address: 44042

```
property dio21_ef_index: int
```

An index to specify the feature you want.

Register: DIO21_EF_INDEX (DIO#(0:22)_EF_INDEX@21)

Address: 44142

```
property dio21_ef_options: int
```

Function dependent on selected feature index.

Register: DIO21_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@21)

Address: 44242

```
property dio21_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO21_EF_READ_A (DIO#(0:22)_EF_READ_A@21)

Address: 3042

```
property dio21_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO21_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@21)

Address: 3142

Type:
Reads the same value as DIO#(0

```
property dio21_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO21_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@21)

Address: 3542

```
property dio21_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO21_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@21)

Address: 3642

```
property dio21_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO21_EF_READ_B (DIO#(0:22)_EF_READ_B@21)

Address: 3242

```
property dio21_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO21_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@21)

Address: 3742

```
property dio22_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO22_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@22)

Address: 44344

```
property dio22_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO22_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@22)

Address: 44444

```
property dio22_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO22_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@22)

Address: 44544

```
property dio22_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO22_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@22)

Address: 44644

```
property dio22_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO22_EF_ENABLE (DIO#(0:22)_EF_ENABLE@22)

Address: 44044

```
property dio22_ef_index: int
```

An index to specify the feature you want.

Register: DIO22_EF_INDEX (DIO#(0:22)_EF_INDEX@22)

Address: 44144

```
property dio22_ef_options: int
```

Function dependent on selected feature index.

Register: DIO22_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@22)

Address: 44244

```
property dio22_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO22_EF_READ_A (DIO#(0:22)_EF_READ_A@22)

Address: 3044

```
property dio22_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO22_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@22)

Address: 3144

Type:
Reads the same value as DIO#(0

```
property dio22_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO22_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@22)

Address: 3544

```
property dio22_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO22_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@22)

Address: 3644

```
property dio22_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO22_EF_READ_B (DIO#(0:22)_EF_READ_B@22)

Address: 3244

```
property dio22_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO22_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@22)

Address: 3744

```
property dio2_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO2_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@2)

Address: 44304

```
property dio2_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO2_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@2)

Address: 44404

```
property dio2_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO2_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@2)

Address: 44504

```
property dio2_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO2_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@2)

Address: 44604

```
property dio2_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO2_EF_ENABLE (DIO#(0:22)_EF_ENABLE@2)

Address: 44004

```
property dio2_ef_index: int
```

An index to specify the feature you want.

Register: DIO2_EF_INDEX (DIO#(0:22)_EF_INDEX@2)

Address: 44104

```
property dio2_ef_options: int
```

Function dependent on selected feature index.

Register: DIO2_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@2)

Address: 44204

```
property dio2_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO2_EF_READ_A (DIO#(0:22)_EF_READ_A@2)

Address: 3004

```
property dio2_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO2_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@2)

Address: 3104

Type:
Reads the same value as DIO#(0

```
property dio2_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO2_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@2)

Address: 3504

```
property dio2_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO2_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@2)

Address: 3604

```
property dio2_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO2_EF_READ_B (DIO#(0:22)_EF_READ_B@2)

Address: 3204

```
property dio2_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO2_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@2)

Address: 3704

```
property dio3_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO3_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@3)

Address: 44306

```
property dio3_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO3_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@3)

Address: 44406

```
property dio3_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO3_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@3)

Address: 44506

```
property dio3_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO3_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@3)

Address: 44606

```
property dio3_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO3_EF_ENABLE (DIO#(0:22)_EF_ENABLE@3)

Address: 44006

```
property dio3_ef_index: int
```

An index to specify the feature you want.

Register: DIO3_EF_INDEX (DIO#(0:22)_EF_INDEX@3)

Address: 44106

```
property dio3_ef_options: int
```

Function dependent on selected feature index.

Register: DIO3_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@3)

Address: 44206

```
property dio3_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO3_EF_READ_A (DIO#(0:22)_EF_READ_A@3)

Address: 3006

```
property dio3_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO3_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@3)

Address: 3106

Type:
Reads the same value as DIO#(0

```
property dio3_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO3_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@3)

Address: 3506

```
property dio3_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO3_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@3)

Address: 3606

```
property dio3_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO3_EF_READ_B (DIO#(0:22)_EF_READ_B@3)

Address: 3206

```
property dio3_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO3_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@3)

Address: 3706

```
property dio4_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO4_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@4)

Address: 44308

```
property dio4_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO4_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@4)

Address: 44408

```
property dio4_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO4_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@4)

Address: 44508

```
property dio4_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO4_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@4)

Address: 44608

```
property dio4_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO4_EF_ENABLE (DIO#(0:22)_EF_ENABLE@4)

Address: 44008

```
property dio4_ef_index: int
```

An index to specify the feature you want.

Register: DIO4_EF_INDEX (DIO#(0:22)_EF_INDEX@4)

Address: 44108

```
property dio4_ef_options: int
```

Function dependent on selected feature index.

Register: DIO4_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@4)

Address: 44208

```
property dio4_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO4_EF_READ_A (DIO#(0:22)_EF_READ_A@4)

Address: 3008

```
property dio4_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO4_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@4)

Address: 3108

Type:
Reads the same value as DIO#(0

```
property dio4_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO4_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@4)

Address: 3508

```
property dio4_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO4_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@4)

Address: 3608

```
property dio4_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO4_EF_READ_B (DIO#(0:22)_EF_READ_B@4)

Address: 3208

```
property dio4_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO4_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@4)

Address: 3708

```
property dio5_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO5_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@5)

Address: 44310

```
property dio5_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO5_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@5)

Address: 44410

```
property dio5_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO5_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@5)

Address: 44510

```
property dio5_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO5_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@5)

Address: 44610

```
property dio5_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO5_EF_ENABLE (DIO#(0:22)_EF_ENABLE@5)

Address: 44010

```
property dio5_ef_index: int
```

An index to specify the feature you want.

Register: DIO5_EF_INDEX (DIO#(0:22)_EF_INDEX@5)

Address: 44110

```
property dio5_ef_options: int
```

Function dependent on selected feature index.

Register: DIO5_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@5)

Address: 44210

```
property dio5_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO5_EF_READ_A (DIO#(0:22)_EF_READ_A@5)

Address: 3010

```
property dio5_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO5_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@5)

Address: 3110

Type:
Reads the same value as DIO#(0

```
property dio5_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO5_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@5)

Address: 3510

```
property dio5_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO5_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@5)

Address: 3610

```
property dio5_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO5_EF_READ_B (DIO#(0:22)_EF_READ_B@5)

Address: 3210

```
property dio5_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO5_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@5)

Address: 3710

```
property dio6_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO6_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@6)

Address: 44312

```
property dio6_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO6_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@6)

Address: 44412

```
property dio6_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO6_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@6)

Address: 44512

```
property dio6_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO6_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@6)

Address: 44612

```
property dio6_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO6_EF_ENABLE (DIO#(0:22)_EF_ENABLE@6)

Address: 44012

```
property dio6_ef_index: int
```

An index to specify the feature you want.

Register: DIO6_EF_INDEX (DIO#(0:22)_EF_INDEX@6)

Address: 44112

```
property dio6_ef_options: int
```

Function dependent on selected feature index.

Register: DIO6_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@6)

Address: 44212

```
property dio6_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO6_EF_READ_A (DIO#(0:22)_EF_READ_A@6)

Address: 3012

```
property dio6_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO6_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@6)

Address: 3112

Type:
Reads the same value as DIO#(0

```
property dio6_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO6_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@6)

Address: 3512

```
property dio6_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO6_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@6)

Address: 3612

```
property dio6_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO6_EF_READ_B (DIO#(0:22)_EF_READ_B@6)

Address: 3212

```
property dio6_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO6_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@6)

Address: 3712

```
property dio7_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO7_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@7)

Address: 44314

```
property dio7_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO7_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@7)

Address: 44414

```
property dio7_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO7_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@7)

Address: 44514

```
property dio7_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO7_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@7)

Address: 44614

```
property dio7_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO7_EF_ENABLE (DIO#(0:22)_EF_ENABLE@7)

Address: 44014

```
property dio7_ef_index: int
```

An index to specify the feature you want.

Register: DIO7_EF_INDEX (DIO#(0:22)_EF_INDEX@7)

Address: 44114

```
property dio7_ef_options: int
```

Function dependent on selected feature index.

Register: DIO7_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@7)

Address: 44214

```
property dio7_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO7_EF_READ_A (DIO#(0:22)_EF_READ_A@7)

Address: 3014

```
property dio7_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO7_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@7)

Address: 3114

Type:
Reads the same value as DIO#(0

```
property dio7_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO7_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@7)

Address: 3514

```
property dio7_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO7_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@7)

Address: 3614

```
property dio7_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO7_EF_READ_B (DIO#(0:22)_EF_READ_B@7)

Address: 3214

```
property dio7_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO7_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@7)

Address: 3714

```
property dio8_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO8_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@8)

Address: 44316

```
property dio8_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO8_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@8)

Address: 44416

```
property dio8_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO8_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@8)

Address: 44516

```
property dio8_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO8_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@8)

Address: 44616

```
property dio8_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO8_EF_ENABLE (DIO#(0:22)_EF_ENABLE@8)

Address: 44016

```
property dio8_ef_index: int
```

An index to specify the feature you want.

Register: DIO8_EF_INDEX (DIO#(0:22)_EF_INDEX@8)

Address: 44116

```
property dio8_ef_options: int
```

Function dependent on selected feature index.

Register: DIO8_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@8)

Address: 44216

```
property dio8_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO8_EF_READ_A (DIO#(0:22)_EF_READ_A@8)

Address: 3016

```
property dio8_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO8_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@8)

Address: 3116

Type:
Reads the same value as DIO#(0

```
property dio8_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO8_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@8)

Address: 3516

```
property dio8_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO8_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@8)

Address: 3616

```
property dio8_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO8_EF_READ_B (DIO#(0:22)_EF_READ_B@8)

Address: 3216

```
property dio8_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO8_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@8)

Address: 3716

```
property dio9_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO9_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@9)

Address: 44318

```
property dio9_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO9_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@9)

Address: 44418

```
property dio9_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO9_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@9)

Address: 44518

```
property dio9_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO9_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@9)

Address: 44618

```
property dio9_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO9_EF_ENABLE (DIO#(0:22)_EF_ENABLE@9)

Address: 44018

```
property dio9_ef_index: int
```

An index to specify the feature you want.

Register: DIO9_EF_INDEX (DIO#(0:22)_EF_INDEX@9)

Address: 44118

```
property dio9_ef_options: int
```

Function dependent on selected feature index.

Register: DIO9_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@9)

Address: 44218

```
property dio9_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO9_EF_READ_A (DIO#(0:22)_EF_READ_A@9)

Address: 3018

```
property dio9_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO9_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@9)

Address: 3118

Type:
Reads the same value as DIO#(0

```
property dio9_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO9_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@9)

Address: 3518

```
property dio9_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO9_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@9)

Address: 3618

```
property dio9_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO9_EF_READ_B (DIO#(0:22)_EF_READ_B@9)

Address: 3218

```
property dio9_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO9_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@9)

Address: 3718

```
property dio_analog_enable: int
```

Read or write the analog configuration of all digital I/O in a single binary-encoded value. 1=Analog mode and 0=Digital mode. When switching from analog to digital, the lines will be set to input. Writes are filtered by the value in DIO_INHIBIT.

Register: DIO_ANALOG_ENABLE

Address: 2880

```
property dio_direction: int
```

Read or write the direction of all digital I/O in a single binary-encoded value. 0=Input and 1=Output. Writes are filtered by the value in DIO_INHIBIT.

Register: DIO_DIRECTION

Address: 2850

```
property dio_ef_clock0_count: int
```

Current tick count of this clock. Will read between 0 and ROLL_VALUE-1.

Register: DIO_EF_CLOCK0_COUNT

Address: 44908

```
property dio_ef_clock0_divisor: int
```

Divides the core clock. Valid options: 1,2,4,8,16,32,64,256.

Register: DIO_EF_CLOCK0_DIVISOR

Address: 44901

```
property dio_ef_clock0_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration.

Register: DIO_EF_CLOCK0_ENABLE

Address: 44900

```
property dio_ef_clock0_options: int
```

Bitmask: bit0: 1 = use external clock. All other bits reserved.

Register: DIO_EF_CLOCK0_OPTIONS

Address: 44902

```
property dio_ef_clock0_roll_value: int
```

The clock will count to this value and then start over at zero. The clock pulses counted are those after the divisor. 0 results in the max roll value possible. This is a 32-bit value (0-4294967295) if using a 32-bit clock, and a 16-bit value (0-65535) if using a 16-bit clock.

Register: DIO_EF_CLOCK0_ROLL_VALUE

Address: 44904

```
property dio_ef_clock1_count: int
```

Current tick count of this clock. Will read between 0 and ROLL_VALUE-1.

Register: DIO_EF_CLOCK1_COUNT

Address: 44918

```
property dio_ef_clock1_divisor: int
```

Divides the core clock. Valid options: 1,2,4,8,16,32,64,256.

Register: DIO_EF_CLOCK1_DIVISOR

Address: 44911

```
property dio_ef_clock1_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration.

Register: DIO_EF_CLOCK1_ENABLE

Address: 44910

```
property dio_ef_clock1_options: int
```

Bitmask: bit0: 1 = use external clock. All other bits reserved.

Register: DIO_EF_CLOCK1_OPTIONS

Address: 44912

```
property dio_ef_clock1_roll_value: int
```

The clock will count to this value and then start over at zero. The clock pulses counted are those after the divisor. 0 results in the max roll value possible. This is a 32-bit value (0-4294967295) if using a 32-bit clock, and a 16-bit value (0-65535) if using a 16-bit clock.

Register: DIO_EF_CLOCK1_ROLL_VALUE

Address: 44914

```
property dio_ef_clock2_count: int
```

Current tick count of this clock. Will read between 0 and ROLL_VALUE-1.

Register: DIO_EF_CLOCK2_COUNT

Address: 44928

```
property dio_ef_clock2_divisor: int
```

Divides the core clock. Valid options: 1,2,4,8,16,32,64,256.

Register: DIO_EF_CLOCK2_DIVISOR

Address: 44921

```
property dio_ef_clock2_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration.

Register: DIO_EF_CLOCK2_ENABLE

Address: 44920

```
property dio_ef_clock2_options: int
```

Bitmask: bit0: 1 = use external clock. All other bits reserved.

Register: DIO_EF_CLOCK2_OPTIONS

Address: 44922

```
property dio_ef_clock2_roll_value: int
```

The clock will count to this value and then start over at zero. The clock pulses counted are those after the divisor. 0 results in the max roll value possible. This is a 32-bit value (0-4294967295) if using a 32-bit clock, and a 16-bit value (0-65535) if using a 16-bit clock.

Register: DIO_EF_CLOCK2_ROLL_VALUE

Address: 44924

```
property dio_inhibit: int
```

A single binary-encoded value where each bit determines whether _STATE, _DIRECTION or _ANALOG_ENABLE writes affect that bit of digital I/O. 0=Default=Affected, 1=Ignored.

Register: DIO_INHIBIT

Address: 2900

```
property dio_pullup_disable: int
```

This register will prevent pullups from being enabled on lines set to digital input. This is a binary coded value where bit 0 represent FIO0 and bit 11 represents EIO3. 1 = pullup disabled, 0 = pullup enabled. This register is not affected by the inhibit register.

Register: DIO_PULLUP_DISABLE

Address: 2890

```
property dio_state: int
```

Read or write the state of all digital I/O in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. A read of an output returns the current logic level on the terminal, not necessarily the output state written. Writes are filtered by the value in DIO_INHIBIT.

Register: DIO_STATE

Address: 2800

```
property eio0: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: EIO0 (EIO#(0:7)@0)

Address: 2008

```
property eio1: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: EIO1 (EIO#(0:7)@1)

Address: 2009

```
property eio2: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: EIO2 (EIO#(0:7)@2)

Address: 2010

```
property eio3: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: EIO3 (EIO#(0:7)@3)

Address: 2011

```
property eio4: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: EIO4 (EIO#(0:7)@4)

Address: 2012

```
property eio5: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: EIO5 (EIO#(0:7)@5)

Address: 2013

```
property eio6: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: EIO6 (EIO#(0:7)@6)

Address: 2014

```
property eio7: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: EIO7 (EIO#(0:7)@7)

Address: 2015

```
property eio_cio_state: int
```

Read or write the state of the 12 bits of EIO-CIO in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. Reading lines set to output returns the current logic levels on the terminals, not necessarily the output states written. As of firmware 1.0172, MIO states are included in the upper nibble of the CIO byte.

Register: EIO_CIO_STATE

Address: 2581

```
property eio_direction: int
```

Read or write the direction of the 8 bits of EIO in a single binary-encoded value. 0=Input and 1=Output. The upper 8-bits of this value are inhibits.

Register: EIO_DIRECTION

Address: 2601

```
property eio_state: int
```

Read or write the state of the 8 bits of EIO in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. Reading lines set to output returns the current logic levels on the terminals, not necessarily the output states written. The upper 8-bits of this value are inhibits.

Register: EIO_STATE

Address: 2501

```
property ethernet_altdns: str
```

Read the current Alt DNS of wired Ethernet.

Register: ETHERNET_ALTDNS

Address: 49108

```
property ethernet_altdns_default: str
```

The Alt DNS of wired Ethernet after a power-cycle to the device.

Register: ETHERNET_ALTDNS_DEFAULT

Address: 49158

```
property ethernet_apply_settings: int
```

Writing 1 to this register power-cycles Ethernet. It tells the device to waits 1s before turning off Ethernet and then 500ms before turning it back on.

Register: ETHERNET_APPLY_SETTINGS

Address: 49190

```
property ethernet_dhcp_enable: int
```

Read the current Enabled/Disabled state of Ethernet DHCP.

Register: ETHERNET_DHCP_ENABLE

Address: 49110

```
property ethernet_dhcp_enable_default: int
```

The Enabled/Disabled state of Ethernet DHCP after a power-cycle to the device.

Register: ETHERNET_DHCP_ENABLE_DEFAULT

Address: 49160

```
property ethernet_dns: str
```

Read the current DNS of wired Ethernet.

Register: ETHERNET_DNS

Address: 49106

```
property ethernet_dns_default: str
```

The DNS of wired Ethernet after a power-cycle to the device.

Register: ETHERNET_DNS_DEFAULT

Address: 49156

```
property ethernet_gateway: str
```

Read the current gateway of wired Ethernet.

Register: ETHERNET_GATEWAY

Address: 49104

```
property ethernet_gateway_default: str
```

The gateway of wired Ethernet after a power-cycle to the device.

Register: ETHERNET_GATEWAY_DEFAULT

Address: 49154

```
property ethernet_ip: str
```

Read the current IP address of wired Ethernet.

Register: ETHERNET_IP

Address: 49100

```
property ethernet_ip_default: str
```

The IP address of wired Ethernet after a power-cycle to the device.

Register: ETHERNET_IP_DEFAULT

Address: 49150

```
property ethernet_mac: str
```

The MAC address of the wired Ethernet module.

Register: ETHERNET_MAC

Address: 60020

```
property ethernet_subnet: str
```

Read the current subnet of wired Ethernet.

Register: ETHERNET_SUBNET

Address: 49102

```
property ethernet_subnet_default: str
```

The subnet of wired Ethernet after a power-cycle to the device.

Register: ETHERNET_SUBNET_DEFAULT

Address: 49152

```
property fio0: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: FIO0 (FIO#(0:7)@0)

Address: 2000

```
property fio1: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: FIO1 (FIO#(0:7)@1)

Address: 2001

```
property fio2: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: FIO2 (FIO#(0:7)@2)

Address: 2002

```
property fio3: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: FIO3 (FIO#(0:7)@3)

Address: 2003

```
property fio4: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: FIO4 (FIO#(0:7)@4)

Address: 2004

```
property fio5: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: FIO5 (FIO#(0:7)@5)

Address: 2005

```
property fio6: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: FIO6 (FIO#(0:7)@6)

Address: 2006

```
property fio7: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: FIO7 (FIO#(0:7)@7)

Address: 2007

```
property fio_direction: int
```

Read or write the direction of the 8 bits of FIO in a single binary-encoded value. 0=Input and 1=Output. The upper 8-bits of this value are inhibits.

Register: FIO_DIRECTION

Address: 2600

```
property fio_eio_state: int
```

Read or write the state of the 16 bits of FIO-EIO in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. Reading lines set to output returns the current logic levels on the terminals, not necessarily the output states written.

Register: FIO_EIO_STATE

Address: 2580

```
property fio_state: int
```

Read or write the state of the 8 bits of FIO in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. Reading lines set to output returns the current logic levels on the terminals, not necessarily the output states written. The upper 8-bits of this value are inhibits.

Register: FIO_STATE

Address: 2500

```
property firmware_version: float
```

The current firmware version installed on the main processor.

Register: FIRMWARE_VERSION

Address: 60004

```
property hardware_installed: int
```

Bitmask indicating installed hardware options. bit0: High Resolution ADC, bit1: WiFi, bit2: RTC, bit3: microSD.

Register: HARDWARE_INSTALLED

Address: 60010

```
property hardware_version: float
```

The hardware version of the device.

Register: HARDWARE_VERSION

Address: 60002

```
property i2c_acks: int
```

A binary encoded value (array of bits) used to observe ACKs from the slave device.

Register: I2C_ACKS

Address: 5114

```
property i2c_data_rx: int
```

Data that has been read from the I2C bus.

This register is a buffer. Underrun behavior - fill with zeros.

Register: I2C_DATA_RX

Address: 5160

```
property i2c_data_tx: int
```

Data that will be written to the I2C bus. This register is a buffer.

Register: I2C_DATA_TX

Address: 5120

```
property i2c_go: int
```

Writing to this register will instruct the LabJack to perform an I2C transaction.

Register: I2C_GO

Address: 5110

```
property i2c_num_bytes_rx: int
```

The number of data bytes to read. Zero is valid and will result in a write-only I2C operation.

Register: I2C_NUM_BYTES_RX

Address: 5109

```
property i2c_num_bytes_tx: int
```

The number of data bytes to transmit. Zero is valid and will result in a read-only I2C operation.

Register: I2C_NUM_BYTES_TX

Address: 5108

```
property i2c_options: int
```

Advanced. Controls details of the I2C protocol to improve device compatibility. bit 0: 1 = Reset the I2C bus before attempting communication. bit 1: 0 = Restarts will use a stop and a start, 1 = Restarts will not use a stop. bit 2: 1 = disable clock stretching.

Register: I2C_OPTIONS

Address: 5103

```
property i2c_scl_dionum: int
```

The number of the DIO line to be used as the I2C clock line. Ex: Writing 1 will force FIO1 to become the I2C-SCL line.

Register: I2C_SCL_DIONUM

Address: 5101

```
property i2c_sda_dionum: int
```

The number of the DIO line to be used as the I2C data line. Ex: Writing 0 will force FIO0 to become the I2C-SDA line.

Register: I2C_SDA_DIONUM

Address: 5100

```
property i2c_slave_address: int
```

The 7-bit address of the slave device. Value is shifted left by firmware to allow room for the I2C R/W bit.

Register: I2C_SLAVE_ADDRESS

Address: 5104

```
property i2c_speed_throttle: int
```

This value controls the I2C clock frequency. Pass 0-65535. Default=0 corresponds to 65536 internally which results in ~450 kHz. 1 results in ~40 Hz, 65516 is ~100 kHz.

Register: I2C_SPEED_THROTTLE

Address: 5102

```
property internal_flash_calculate_crc: int
```

Calculates the checksum of a 4096 byte flash page.

Register: INTERNAL_FLASH_CALCULATE_CRC

Address: 61842

```
property internal_flash_erase: int
```

Erases a 4k section of internal flash starting at the specified address. This register is a buffer.

Register: INTERNAL_FLASH_ERASE

Address: 61820

```
property internal_flash_key: int
```

Sets the region of internal flash to which access is allowed.

Register: INTERNAL_FLASH_KEY

Address: 61800

```
property internal_flash_read: int
```

Data read from internal flash. Read size must be an even number of registers.

Register: INTERNAL_FLASH_READ

Address: 61812

```
property internal_flash_read_pointer: int
```

The address in internal flash that reads will start from.

Register: INTERNAL_FLASH_READ_POINTER

Address: 61810

```
property internal_flash_write: int
```

Data written here will be written to internal flash. Write size must be an even number of registers. This register is a buffer.

Register: INTERNAL_FLASH_WRITE

Address: 61832

```
property internal_flash_write_pointer: int
```

Address in internal flash where writes will begin.

Register: INTERNAL_FLASH_WRITE_POINTER

Address: 61830

```
property io_config_check_for_default: int
```

Returns 1 if the current configuration matches the default configuration.

Register: IO_CONFIG_CHECK_FOR_DEFAULT

Address: 49018

```
property io_config_check_for_factory: int
```

Returns 1 if default settings are the same as factory settings.

Register: IO_CONFIG_CHECK_FOR_FACTORY

Address: 49000

```
property io_config_current_crc32: int
```

Collects the current IO configuration and calculates of a CRC32.

Register: IO_CONFIG_CURRENT_CRC32

Address: 49020

```
property io_config_current_pread: int
```

The address in the current configuration data which reads will start from.

Register: IO_CONFIG_CURRENT_pREAD

Address: 49014

```
property io_config_current_read: int
```

Data read from the current configuration.

Register: IO_CONFIG_CURRENT_READ

Address: 49016

```
property io_config_default_pread: int
```

The address in the configuration data which reads will start from.

Register: IO_CONFIG_DEFAULT_pREAD

Address: 49010

```
property io_config_default_read: int
```

Data read from the startup configuration.

Register: IO_CONFIG_DEFAULT_READ

Address: 49012

```
property io_config_factory_pread: int
```

The address in the factory configuration data which reads will start from.

Register: IO_CONFIG_FACTORY_pREAD

Address: 49006

```
property io_config_factory_read: int
```

Data read from factory configuration.

Register: IO_CONFIG_FACTORY_READ

Address: 49008

```
property io_config_set_current_to_default: int
```

Write a 1 to set current values to default configuration. The default values are retrieved from flash and written to the current configuration registers, thus this behaves similar to reboot/power-up. Systems affected: AIN, DIO, DAC, AIN_EF, DIO_EF.

Register: IO_CONFIG_SET_CURRENT_TO_DEFAULT

Address: 61991

```
property io_config_set_current_to_factory: int
```

Write a 1 to set current values to factory configuration. The factory values are retrieved from flash and written to the current configuration registers. Systems affected: AIN, DIO, DAC, AIN_EF, DIO_EF.

Register: IO_CONFIG_SET_CURRENT_TO_FACTORY

Address: 61990

```
property io_config_set_default_to_current: int
```

Write a 1 to cause new default (reboot/power-up) values to be saved to flash. Current values are retrieved and saved as the new defaults. Systems affected: AIN, DIO, DAC, AIN_EF, DIO_EF.

Register: IO_CONFIG_SET_DEFAULT_TO_CURRENT

Address: 49002

```
property io_config_set_default_to_factory: int
```

Write a 1 to cause new default (reboot/power-up) values to be saved to flash. Factory values are retrieved and saved as the new defaults. Systems affected: AIN, DIO, DAC, AIN_EF, DIO_EF.

Register: IO_CONFIG_SET_DEFAULT_TO_FACTORY

Address: 49004

```
property last_err_detail: int
```

Returns the last LabJack error code seen on the device.

Register: LAST_ERR_DETAIL

Address: 55000

```
property last_err_frame: int
```

Returns the frame number that caused the last error.

Register: LAST_ERR_FRAME

Address: 55002

```
property last_err_transaction_id: int
```

Returns the transaction ID of the Modbus TCP packet that caused the last error.

Register: LAST_ERR_TRANSACTION_ID

Address: 55003

```
property last_mb_err: int
```

Returns the last Modbus TCP error code seen on the device.

Register: LAST_MB_ERR

Address: 55001

```
property led_comm: int
```

Sets the state of the COMM LED when the LEDs are set to manual, see the POWER_LED register.

Register: LED_COMM

Address: 2990

```
property led_status: int
```

Sets the state of the STATUS LED when the LEDs are set to manual, see the POWER_LED register.

Register: LED_STATUS

Address: 2991

```
property lua_debug_data: int
```

Read debug data from here. This register is a buffer.

Register: LUA_DEBUG_DATA

Address: 6024

```
property lua_debug_enable: int
```

Write 1 to this register to enable debugging.

Register: LUA_DEBUG_ENABLE

Address: 6020

```
property lua_debug_enable_default: int
```

This is the value that will be loaded into LUA_DEBUG_ENABLE when the LabJack boots up.

Register: LUA_DEBUG_ENABLE_DEFAULT

Address: 6120

```
property lua_debug_num_bytes: int
```

The number of data bytes in the debug buffer waiting to be read.

Register: LUA_DEBUG_NUM_BYTES

Address: 6022

```
property lua_debug_num_bytes_default: int
```

This is the number of bytes that will be used for the lua debug buffer.

Register: LUA_DEBUG_NUM_BYTES_DEFAULT

Address: 6122

```
property lua_load_saved: int
```

Writing any value reads the script from Flash and loads it into RAM. The script can now be compiled and run.

Register: LUA_LOAD_SAVED

Address: 6034

```
property lua_no_warn_truncation: int
```

Writing a 1 to this register will prevent truncation warnings from being displayed. This register will be cleared every time Lua is started.

Register: LUA_NO_WARN_TRUNCATION

Address: 6050

```
property lua_run: int
```

Writing 1 compiles and runs the Lua script that is loaded in RAM. Writing zero stops the script and begins cleaning up memory. Users may poll the register after writing a value of 0 to verify that the VM is unloaded, and garbage collection is complete. 0 = VM fully unloaded. 1 = Running/in-progress.

Register: LUA_RUN

Address: 6000

```
property lua_run_default: int
```

If set to 1 the script saved in flash will be loaded and run when the LabJack boots up.

Register: LUA_RUN_DEFAULT

Address: 6100

```
property lua_save_to_flash: int
```

Write 1 to save the loaded source code to flash.

Register: LUA_SAVE_TO_FLASH

Address: 6032

```
property lua_saved_read: int
```

Read script saved to flash from here.

Register: LUA_SAVED_READ

Address: 6038

```
property lua_saved_read_pointer: int
```

Address within the saved Lua script in Flash to begin reading from.

Register: LUA_SAVED_READ_POINTER

Address: 6036

```
property lua_source_size: int
```

Allocates RAM for the source code.

Register: LUA_SOURCE_SIZE

Address: 6012

```
property lua_source_write: int
```

Write the source code here. Source will be saved to the RAM allocated by LUA_SOURCE_SIZE. This register is a buffer.

Register: LUA_SOURCE_WRITE

Address: 6014

```
property mio_direction: int
```

Read or write the direction of the 3 bits of MIO in a single binary-encoded value. 0=Input and 1=Output. The upper 8-bits of this value are inhibits.

Register: MIO_DIRECTION

Address: 2603

```
property mio_state: int
```

Read or write the state of the 3 bits of MIO in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. Reading lines set to output returns the current logic levels on the terminals, not necessarily the output states written. The upper 8-bits of this value are inhibits.

Register: MIO_STATE

Address: 2503

```
property onewire_data_rx: int
```

Data received over the 1-wire bus.

This register is a buffer. Underrun behavior - buffer is static, old data will fill the extra locations, firmware 1.0225 changes this to read zeros.

Register: ONEWIRE_DATA_RX

Address: 5370

```
property onewire_data_tx: int
```

Data to be transmitted over the 1-wire bus. This register is a buffer.

Register: ONEWIRE_DATA_TX

Address: 5340

```
property onewire_dpu_dionum: int
```

The dynamic pullup control DIO number.

Register: ONEWIRE_DPU_DIONUM

Address: 5301

```
property onewire_dq_dionum: int
```

The data-line DIO number.

Register: ONEWIRE_DQ_DIONUM

Address: 5300

```
property onewire_function: int
```

Set the ROM function to use. 0xF0=Search, 0xCC=Skip, 0x55=Match, 0x33=Read.

Register: ONEWIRE_FUNCTION

Address: 5307

```
property onewire_go: int
```

Instructs the device to perform the configured 1-wire transaction.

Register: ONEWIRE_GO

Address: 5310

```
property onewire_num_bytes_rx: int
```

Number of data bytes to be received.

Register: ONEWIRE_NUM_BYTES_RX

Address: 5309

```
property onewire_num_bytes_tx: int
```

Number of data bytes to be sent.

Register: ONEWIRE_NUM_BYTES_TX

Address: 5308

```
property onewire_options: int
```

Controls advanced features. Value is a bitmask. bit 0: reserved, bit 1: reserved, bit 2: 1=DPU Enabled 0=DPU Disabled, bit 3: DPU Polarity 1=Active state is high, 0=Active state is low (Dynamic Pull-Up)

Register: ONEWIRE_OPTIONS

Address: 5302

```
property onewire_path_h: int
```

Upper 32-bits of the path to take during a search.

Register: ONEWIRE_PATH_H

Address: 5324

```
property onewire_path_l: int
```

Lower 32-bits of the path to take during a search.

Register: ONEWIRE_PATH_L

Address: 5326

```
property onewire_rom_branchs_found_h: int
```

Upper 32-bits of the branches detected during a search.

Register: ONEWIRE_ROM_BRANCHS_FOUND_H

Address: 5332

```
property onewire_rom_branchs_found_l: int
```

Lower 32-bits of the branches detected during a search.

Register: ONEWIRE_ROM_BRANCHS_FOUND_L

Address: 5334

```
property onewire_rom_match_h: int
```

Upper 32-bits of the ROM to match.

Register: ONEWIRE_ROM_MATCH_H

Address: 5320

```
property onewire_rom_match_l: int
```

Lower 32-bits of the ROM to match.

Register: ONEWIRE_ROM_MATCH_L

Address: 5322

```
property onewire_search_result_h: int
```

Upper 32-bits of the search result.

Register: ONEWIRE_SEARCH_RESULT_H

Address: 5328

```
property onewire_search_result_l: int
```

Lower 32-bites of the search result.

Register: ONEWIRE_SEARCH_RESULT_L

Address: 5330

```
property power_core: int
```

The core processor speed. 0=80MHz, 1=20MHz, 2=2MHz, 3=250kHz.

Register: POWER_CORE

Address: 48001

```
property power_core_default: int
```

Controls the core processor speed after a power-cycle to the device. 0=80MHz, 1=20MHz, 2=2MHz, 3=250kHz.

Register: POWER_CORE_DEFAULT

Address: 48051

```
property power_ethernet: int
```

The current ON/OFF state of the Ethernet module.

Register: POWER_ETHERNET

Address: 48003

```
property power_ethernet_default: int
```

The ON/OFF state of the Ethernet module after a power-cycle to the device. Provided to optionally reduce power consumption.

Register: POWER_ETHERNET_DEFAULT

Address: 48053

```
property power_led: int
```

Sets the LED operation: 0 = Off. Useful for lower power applications. 1 = normal. 2 = Lower power, LEDs will still blink but will normally be off. 3 = Reserved. 4 = Manual, in this mode the LEDs can be user controlled.

Register: POWER_LED

Address: 48006

```
property power_led_default: int
```

The ON/OFF state of the LEDs after a power-cycle to the device.

Register: POWER_LED_DEFAULT

Address: 48056

```
property power_usb: int
```

The current ON/OFF state of the USB module.

Register: POWER_USB

Address: 48002

```
property power_usb_default: int
```

The current ON/OFF state of the USB module after a power-cycle to the device.

Register: POWER_USB_DEFAULT

Address: 48052

```
property product_id: float
```

The numeric identifier of the device. Such as 7 for a T7 / T7-Pro.

Register: PRODUCT_ID

Address: 60000

```
property rtc_set_time_s: int
```

Write a new timestamp to the RTC in seconds since Jan, 1970, aka Epoch or Unix timestamp.

Register: RTC_SET_TIME_S

Address: 61504

```
property rtc_set_time_sntp: int
```

Write any value to instruct the T7 to update its clock from a SNTP server. Requires that SNTP_UPDATE_INTERVAL is non-zero.

Register: RTC_SET_TIME_SNTP

Address: 61506

```
property rtc_time_calendar: int
```

Read six consecutive addresses of type UINT16, starting with this address. The result will be in year, month, day, hour, minute, second calendar format. i.e. [2014, 10, 21, 18, 55, 35]

Register: RTC_TIME_CALENDAR

Address: 61510

```
property rtc_time_s: int
```

Read the current time in seconds since Jan, 1970, aka Epoch or Unix time. This value is calculated from the 80 MHz crystal, not the RTC 32 kHz crystal. Non-pro devices do not have a real time clock, so the reported time is either seconds since device startup or the time reported by the network time protocol over Ethernet. Pro devices have a real time clock that will be used to initialize this register at startup, the time can then be updated by NTP.

Register: RTC_TIME_S

Address: 61500

```
property sbus0_background_enable: int
```

Currently unsupported.

Register: SBUS0_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@0)

Address: 30250

```
property sbus0_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS0_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@0)

Address: 30225

```
property sbus0_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS0_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@0)

Address: 30200

```
property sbus0_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS0_RH (SBUS#(0:22)_RH@0)

Address: 30150

```
property sbus0_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS0_TEMP (SBUS#(0:22)_TEMP@0)

Address: 30100

```
property sbus10_background_enable: int
```

Currently unsupported.

Register: SBUS10_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@10)

Address: 30260

```
property sbus10_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS10_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@10)

Address: 30235

```
property sbus10_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS10_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@10)

Address: 30210

```
property sbus10_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS10_RH (SBUS#(0:22)_RH@10)

Address: 30170

```
property sbus10_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS10_TEMP (SBUS#(0:22)_TEMP@10)

Address: 30120

```
property sbus11_background_enable: int
```

Currently unsupported.

Register: SBUS11_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@11)

Address: 30261

```
property sbus11_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS11_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@11)

Address: 30236

```
property sbus11_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS11_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@11)

Address: 30211

```
property sbus11_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS11_RH (SBUS#(0:22)_RH@11)

Address: 30172

```
property sbus11_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS11_TEMP (SBUS#(0:22)_TEMP@11)

Address: 30122

```
property sbus12_background_enable: int
```

Currently unsupported.

Register: SBUS12_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@12)

Address: 30262

```
property sbus12_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS12_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@12)

Address: 30237

```
property sbus12_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS12_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@12)

Address: 30212

```
property sbus12_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS12_RH (SBUS#(0:22)_RH@12)

Address: 30174

```
property sbus12_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS12_TEMP (SBUS#(0:22)_TEMP@12)

Address: 30124

```
property sbus13_background_enable: int
```

Currently unsupported.

Register: SBUS13_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@13)

Address: 30263

```
property sbus13_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS13_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@13)

Address: 30238

```
property sbus13_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS13_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@13)

Address: 30213

```
property sbus13_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS13_RH (SBUS#(0:22)_RH@13)

Address: 30176

```
property sbus13_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS13_TEMP (SBUS#(0:22)_TEMP@13)

Address: 30126

```
property sbus14_background_enable: int
```

Currently unsupported.

Register: SBUS14_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@14)

Address: 30264

```
property sbus14_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS14_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@14)

Address: 30239

```
property sbus14_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS14_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@14)

Address: 30214

```
property sbus14_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS14_RH (SBUS#(0:22)_RH@14)

Address: 30178

```
property sbus14_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS14_TEMP (SBUS#(0:22)_TEMP@14)

Address: 30128

```
property sbus15_background_enable: int
```

Currently unsupported.

Register: SBUS15_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@15)

Address: 30265

```
property sbus15_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS15_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@15)

Address: 30240

```
property sbus15_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS15_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@15)

Address: 30215

```
property sbus15_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS15_RH (SBUS#(0:22)_RH@15)

Address: 30180

```
property sbus15_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS15_TEMP (SBUS#(0:22)_TEMP@15)

Address: 30130

```
property sbus16_background_enable: int
```

Currently unsupported.

Register: SBUS16_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@16)

Address: 30266

```
property sbus16_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS16_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@16)

Address: 30241

```
property sbus16_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS16_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@16)

Address: 30216

```
property sbus16_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS16_RH (SBUS#(0:22)_RH@16)

Address: 30182

```
property sbus16_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS16_TEMP (SBUS#(0:22)_TEMP@16)

Address: 30132

```
property sbus17_background_enable: int
```

Currently unsupported.

Register: SBUS17_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@17)

Address: 30267

```
property sbus17_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS17_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@17)

Address: 30242

```
property sbus17_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS17_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@17)

Address: 30217

```
property sbus17_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS17_RH (SBUS#(0:22)_RH@17)

Address: 30184

```
property sbus17_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS17_TEMP (SBUS#(0:22)_TEMP@17)

Address: 30134

```
property sbus18_background_enable: int
```

Currently unsupported.

Register: SBUS18_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@18)

Address: 30268

```
property sbus18_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS18_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@18)

Address: 30243

```
property sbus18_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS18_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@18)

Address: 30218

```
property sbus18_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS18_RH (SBUS#(0:22)_RH@18)

Address: 30186

```
property sbus18_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS18_TEMP (SBUS#(0:22)_TEMP@18)

Address: 30136

```
property sbus19_background_enable: int
```

Currently unsupported.

Register: SBUS19_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@19)

Address: 30269

```
property sbus19_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS19_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@19)

Address: 30244

```
property sbus19_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS19_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@19)

Address: 30219

```
property sbus19_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS19_RH (SBUS#(0:22)_RH@19)

Address: 30188

```
property sbus19_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS19_TEMP (SBUS#(0:22)_TEMP@19)

Address: 30138

```
property sbus1_background_enable: int
```

Currently unsupported.

Register: SBUS1_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@1)

Address: 30251

```
property sbus1_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS1_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@1)

Address: 30226

```
property sbus1_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS1_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@1)

Address: 30201

```
property sbus1_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS1_RH (SBUS#(0:22)_RH@1)

Address: 30152

```
property sbus1_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS1_TEMP (SBUS#(0:22)_TEMP@1)

Address: 30102

```
property sbus20_background_enable: int
```

Currently unsupported.

Register: SBUS20_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@20)

Address: 30270

```
property sbus20_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS20_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@20)

Address: 30245

```
property sbus20_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS20_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@20)

Address: 30220

```
property sbus20_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS20_RH (SBUS#(0:22)_RH@20)

Address: 30190

```
property sbus20_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS20_TEMP (SBUS#(0:22)_TEMP@20)

Address: 30140

```
property sbus21_background_enable: int
```

Currently unsupported.

Register: SBUS21_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@21)

Address: 30271

```
property sbus21_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS21_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@21)

Address: 30246

```
property sbus21_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS21_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@21)

Address: 30221

```
property sbus21_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS21_RH (SBUS#(0:22)_RH@21)

Address: 30192

```
property sbus21_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS21_TEMP (SBUS#(0:22)_TEMP@21)

Address: 30142

```
property sbus22_background_enable: int
```

Currently unsupported.

Register: SBUS22_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@22)

Address: 30272

```
property sbus22_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS22_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@22)

Address: 30247

```
property sbus22_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS22_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@22)

Address: 30222

```
property sbus22_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS22_RH (SBUS#(0:22)_RH@22)

Address: 30194

```
property sbus22_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS22_TEMP (SBUS#(0:22)_TEMP@22)

Address: 30144

```
property sbus2_background_enable: int
```

Currently unsupported.

Register: SBUS2_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@2)

Address: 30252

```
property sbus2_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS2_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@2)

Address: 30227

```
property sbus2_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS2_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@2)

Address: 30202

```
property sbus2_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS2_RH (SBUS#(0:22)_RH@2)

Address: 30154

```
property sbus2_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS2_TEMP (SBUS#(0:22)_TEMP@2)

Address: 30104

```
property sbus3_background_enable: int
```

Currently unsupported.

Register: SBUS3_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@3)

Address: 30253

```
property sbus3_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS3_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@3)

Address: 30228

```
property sbus3_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS3_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@3)

Address: 30203

```
property sbus3_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS3_RH (SBUS#(0:22)_RH@3)

Address: 30156

```
property sbus3_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS3_TEMP (SBUS#(0:22)_TEMP@3)

Address: 30106

```
property sbus4_background_enable: int
```

Currently unsupported.

Register: SBUS4_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@4)

Address: 30254

```
property sbus4_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS4_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@4)

Address: 30229

```
property sbus4_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS4_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@4)

Address: 30204

```
property sbus4_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS4_RH (SBUS#(0:22)_RH@4)

Address: 30158

```
property sbus4_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS4_TEMP (SBUS#(0:22)_TEMP@4)

Address: 30108

```
property sbus5_background_enable: int
```

Currently unsupported.

Register: SBUS5_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@5)

Address: 30255

```
property sbus5_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS5_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@5)

Address: 30230

```
property sbus5_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS5_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@5)

Address: 30205

```
property sbus5_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS5_RH (SBUS#(0:22)_RH@5)

Address: 30160

```
property sbus5_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS5_TEMP (SBUS#(0:22)_TEMP@5)

Address: 30110

```
property sbus6_background_enable: int
```

Currently unsupported.

Register: SBUS6_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@6)

Address: 30256

```
property sbus6_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS6_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@6)

Address: 30231

```
property sbus6_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS6_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@6)

Address: 30206

```
property sbus6_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS6_RH (SBUS#(0:22)_RH@6)

Address: 30162

```
property sbus6_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS6_TEMP (SBUS#(0:22)_TEMP@6)

Address: 30112

```
property sbus7_background_enable: int
```

Currently unsupported.

Register: SBUS7_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@7)

Address: 30257

```
property sbus7_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS7_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@7)

Address: 30232

```
property sbus7_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS7_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@7)

Address: 30207

```
property sbus7_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS7_RH (SBUS#(0:22)_RH@7)

Address: 30164

```
property sbus7_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS7_TEMP (SBUS#(0:22)_TEMP@7)

Address: 30114

```
property sbus8_background_enable: int
```

Currently unsupported.

Register: SBUS8_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@8)

Address: 30258

```
property sbus8_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS8_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@8)

Address: 30233

```
property sbus8_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS8_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@8)

Address: 30208

```
property sbus8_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS8_RH (SBUS#(0:22)_RH@8)

Address: 30166

```
property sbus8_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS8_TEMP (SBUS#(0:22)_TEMP@8)

Address: 30116

```
property sbus9_background_enable: int
```

Currently unsupported.

Register: SBUS9_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@9)

Address: 30259

```
property sbus9_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS9_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@9)

Address: 30234

```
property sbus9_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS9_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@9)

Address: 30209

```
property sbus9_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS9_RH (SBUS#(0:22)_RH@9)

Address: 30168

```
property sbus9_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS9_TEMP (SBUS#(0:22)_TEMP@9)

Address: 30118

```
property sbus_all_clock_dionum: int
```

A write to this global parameter sets all SBUS clock line registers to the same value. A read will return the correct setting if all channels are set the same, but otherwise will return 0xFF.

Register: SBUS_ALL_CLOCK_DIONUM

Address: 30276

```
property sbus_all_clock_speed: int
```

Sets the clock speed. The clock is software generated so the resulting frequency is not exact. Valid range is 0-65535. Larger values are faster. 0 is the fastest option and is equivalent to 65536. A value of 0 is ~200 kHz. A value of 65000 is ~9.1 kHz.

Register: SBUS_ALL_CLOCK_SPEED

Address: 30278

```
property sbus_all_data_dionum: int
```

A write to this global parameter sets all SBUS data line registers to the same value. A read will return the correct setting if all channels are set the same, but otherwise will return 0xFF.

Register: SBUS_ALL_DATA_DIONUM

Address: 30275

```
property sbus_all_power_dionum: int
```

Sets the power line. This DIO is set to output-high upon any read of SBUS#_TEMP or SBUS#_RH. Default is FIO6 for the T4 and FIO2 for the T7. An FIO line can power up to 4 sensors while an EIO/CIO/MIO line or DAC line can power up to 20 sensors. Set to 9999 to disable. To use multiple power lines, use a DAC line for power, or otherwise control power yourself, set this to 9999 and then control power using writes to normal registers such as FIO5, EIO0, or DAC0.

Register: SBUS_ALL_POWER_DIONUM

Address: 30277

```
property serial_number: int
```

The serial number of the device.

Register: SERIAL_NUMBER

Address: 60028

```
setup_ain0(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN0.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain1(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN1.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain10(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN10.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain11(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN11.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain2(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN2.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain3(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN3.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain4(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN4.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain5(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN5.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain6(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN6.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain7(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN7.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain8(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN8.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain9(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN9.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain_all(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup all AIN channels.

```
property sntp_update_interval: int
```

Sets the SNTP retry time in seconds. A value of zero will disable SNTP(0=default). Values must be 10 or greater.

Register: SNTP_UPDATE_INTERVAL

Address: 49702

```
property spc_frequency_out_enable_deprecated: int
```

This has been deprecated in favor of DAC1_FREQUENCY_OUT_ENABLE. 0 = off, 1 = output 10 Hz signal on SPC. Note that stream uses SPC for a timing output, so if you use this while streaming you will get a lot more counts.

Register: SPC_FREQUENCY_OUT_ENABLE_DEPRECATED

Address: 61530

```
property spi_clk_dionum: int
```

The DIO line for Clock.

Register: SPI_CLK_DIONUM

Address: 5001

```
property spi_cs_dionum: int
```

The DIO line for Chip-Select.

Register: SPI_CS_DIONUM

Address: 5000

```
property spi_data_rx: int
```

Read data here.

This register is a buffer. Underrun behavior - fill with zeros.

Register: SPI_DATA_RX

Address: 5050

```
property spi_data_tx: int
```

Write data here. This register is a buffer.

Register: SPI_DATA_TX

Address: 5010

```
property spi_go: int
```

Write 1 to begin the configured SPI transaction.

Register: SPI_GO

Address: 5007

```
property spi_miso_dionum: int
```

The DIO line for Master-In-Slave-Out.

Register: SPI_MISO_DIONUM

Address: 5002

```
property spi_mode: int
```

The SPI mode controls the clock idle state and which edge clocks the data. Bit 1 is CPOL and Bit 0 is CPHA, so CPOL/CPHA for different decimal values: 0 = 0/0 = b00, 1 = 0/1 = b01, 2 = 1/0 = b10, 3 = 1/1 = b11. For CPOL and CPHA explanations, see Wikipedia article: https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus.

Register: SPI_MODE

Address: 5004

```
property spi_mosi_dionum: int
```

The DIO line for Master-Out-Slave-In.

Register: SPI_MOSI_DIONUM

Address: 5003

```
property spi_num_bytes: int
```

The number of bytes to transfer. The maximum transfer size is 100 bytes.

Register: SPI_NUM_BYTES

Address: 5009

```
property spi_options: int
```

Bit 0 is Auto-CS-Disable. When bit 0 is 0, CS is enabled. When bit 0 is 1, CS is disabled. Bit 1: 0 = Set DIO directions before starting the SPI operations, 1 = Do not set DIO directions. Bit 2: 0 = Transmit data MSB first, 1 = LSB first. Bits 4-7: This value sets the number of bits that will be transmitted during the last byte of the SPI operation. Default is 8, valid options are 1-8.

Register: SPI_OPTIONS

Address: 5006

```
property spi_speed_throttle: int
```

This value controls the SPI clock frequency. Pass 0-65535. Default=0 corresponds to 65536 internally which results in ~800 kHz. 65500 = ~100 kHz, 65100 = ~10 kHz, 61100 = ~1 kHz, 21000 = ~100 Hz, and 1 = ~67 Hz. Avoid setting too low such that the entire transaction lasts longer than the 250 millisecond timeout of the internal watchdog timer.

Register: SPI_SPEED_THROTTLE

Address: 5005

```
property stream_auto_target: int
```

Controls where data will be sent. Value is a bitmask. bit 0: 1 = Send to Ethernet 702 sockets, bit 1: 1 = Send to USB, bit 4: 1 = Command-Response mode. All other bits are reserved.

Register: STREAM_AUTO_TARGET

Address: 4016

```
property stream_buffer_size_bytes: int
```

Size of the stream data buffer in bytes. A value of 0 equates to the default value. Must be a power of 2. Size in samples is STREAM_BUFFER_SIZE_BYTES/2. Size in scans is (STREAM_BUFFER_SIZE_BYTES/2)/STREAM_NUM_ADDRESSES. Changes while stream is running do not affect the currently running stream.

Register: STREAM_BUFFER_SIZE_BYTES

Address: 4012

```
property stream_data_capture_16: int
```

If a channel produces 32-bits of data the upper 16 will be saved here.

Register: STREAM_DATA_CAPTURE_16

Address: 4899

```
property stream_data_cr: int
```

Address to read stream data from when operating in C-R mode.

Register: STREAM_DATA_CR

Address: 4500

```
property stream_enable: int
```

Write 1 to start stream. Write 0 to stop stream. Reading this register returns 1 when stream is enabled. When using a triggered stream the stream is considered enabled while waiting for the trigger.

Register: STREAM_ENABLE

Address: 4990

```
property stream_num_addresses: int
```

The number of entries in the scanlist.

Register: STREAM_NUM_ADDRESSES

Address: 4004

```
property stream_num_scans: int
```

The number of scans to run before automatically stopping (stream-burst). 0 = run continuously. Limit for STREAM_NUM_SCANS is 2^32-1, but if the host is not reading data as fast as it is acquired you also need to consider STREAM_BUFFER_SIZE_BYTES.

Register: STREAM_NUM_SCANS

Address: 4020

```
property stream_out0: int
```

127) to trigger stream-out updates. When added to the scan list these do count against the max scan rate just like normal input addresses, but they do not return any data in the stream read.

Register: STREAM_OUT0 (STREAM_OUT#(0:3)@0)

Address: 4800

Type:
Include one or more of these registers in STREAM_SCANLIST_ADDRESS#(0

```
property stream_out0_buffer_allocate_num_bytes: int
```

Size of the buffer in bytes as a power of 2. Should be at least twice the size of updates that will be written and no less than 32. The usable buffer size will be equal to the value of this register minus 2 bytes (one 16-bit sample). Before writing data to _BUFFER_###, you must write to _BUFFER_ALLOCATE_NUM_BYTES to allocate RAM for the data. Max is 16384.

Register: STREAM_OUT0_BUFFER_ALLOCATE_NUM_BYTES
(STREAM_OUT#(0:3)_BUFFER_ALLOCATE_NUM_BYTES@0)

Address: 4050

```
property stream_out0_buffer_f32: float
```

Data destination when sending floating point data. Appropriate cal constants are used to convert F32 values to 16-bit binary data, and thus each of these values uses 2 bytes of the stream-out buffer. This register is a buffer.

Register: STREAM_OUT0_BUFFER_F32 (STREAM_OUT#(0:3)_BUFFER_F32@0)

Address: 4400

```
property stream_out0_buffer_status: int
```

The number of values in the buffer that are not currently being used.

Register: STREAM_OUT0_BUFFER_STATUS (STREAM_OUT#(0:3)_BUFFER_STATUS@0)

Address: 4080

```
property stream_out0_buffer_u16: int
```

Data destination when sending 16-bit integer data. Each value uses 2 bytes of the stream-out buffer. This register is a buffer.

Register: STREAM_OUT0_BUFFER_U16 (STREAM_OUT#(0:3)_BUFFER_U16@0)

Address: 4420

```
property stream_out0_buffer_u32: int
```

Not used at this time. There are no U32 registers supported in stream-out. This register is a buffer.

Register: STREAM_OUT0_BUFFER_U32 (STREAM_OUT#(0:3)_BUFFER_U32@0)

Address: 4410

```
property stream_out0_enable: int
```

When STREAM_OUT#_ENABLE is enabled, the stream out target is generally updated by one value from the stream out buffer per stream scan. For example, there will generally be one instance of e.g. STREAM_OUT0 in the stream scan list, which will cause one STREAM_OUT0_BUFFER value to be consumed and written to STREAM_OUT0_TARGET for every stream scan. The stream scan list could also contain two instances of STREAM_OUT0, in which case two values from STREAM_OUT0_BUFFER value would be consumed and written for every stream scan.

Register: STREAM_OUT0_ENABLE (STREAM_OUT#(0:3)_ENABLE@0)

Address: 4090

```
property stream_out0_loop_num_values: int
```

The number of values, from the end of the array, that will be repeated after reaching the end of supplied data array.

Register: STREAM_OUT0_LOOP_NUM_VALUES (STREAM_OUT#(0:3)_LOOP_NUM_VALUES@0)

Address: 4060

```
property stream_out0_set_loop: int
```

Controls when new data and loop size are used. 1=Use new data immediately. 2=Wait for synch. New data will not be used until a different stream-out channel is set to Synch. 3=Synch. This stream-out# as well as any stream-outs set to synch will start using new data immediately.

Register: STREAM_OUT0_SET_LOOP (STREAM_OUT#(0:3)_SET_LOOP@0)

Address: 4070

```
property stream_out0_target: int
```

Channel that data will be written to. Before writing data to _BUFFER_###, you must write to _TARGET so the device knows how to interpret and store values.

Register: STREAM_OUT0_TARGET (STREAM_OUT#(0:3)_TARGET@0)

Address: 4040

```
property stream_out1: int
```

127) to trigger stream-out updates. When added to the scan list these do count against the max scan rate just like normal input addresses, but they do not return any data in the stream read.

Register: STREAM_OUT1 (STREAM_OUT#(0:3)@1)

Address: 4801

Type:
Include one or more of these registers in STREAM_SCANLIST_ADDRESS#(0

```
property stream_out1_buffer_allocate_num_bytes: int
```

Size of the buffer in bytes as a power of 2. Should be at least twice the size of updates that will be written and no less than 32. The usable buffer size will be equal to the value of this register minus 2 bytes (one 16-bit sample). Before writing data to _BUFFER_###, you must write to _BUFFER_ALLOCATE_NUM_BYTES to allocate RAM for the data. Max is 16384.

Register: STREAM_OUT1_BUFFER_ALLOCATE_NUM_BYTES
(STREAM_OUT#(0:3)_BUFFER_ALLOCATE_NUM_BYTES@1)

Address: 4052

```
property stream_out1_buffer_f32: float
```

Data destination when sending floating point data. Appropriate cal constants are used to convert F32 values to 16-bit binary data, and thus each of these values uses 2 bytes of the stream-out buffer. This register is a buffer.

Register: STREAM_OUT1_BUFFER_F32 (STREAM_OUT#(0:3)_BUFFER_F32@1)

Address: 4402

```
property stream_out1_buffer_status: int
```

The number of values in the buffer that are not currently being used.

Register: STREAM_OUT1_BUFFER_STATUS (STREAM_OUT#(0:3)_BUFFER_STATUS@1)

Address: 4082

```
property stream_out1_buffer_u16: int
```

Data destination when sending 16-bit integer data. Each value uses 2 bytes of the stream-out buffer. This register is a buffer.

Register: STREAM_OUT1_BUFFER_U16 (STREAM_OUT#(0:3)_BUFFER_U16@1)

Address: 4421

```
property stream_out1_buffer_u32: int
```

Not used at this time. There are no U32 registers supported in stream-out. This register is a buffer.

Register: STREAM_OUT1_BUFFER_U32 (STREAM_OUT#(0:3)_BUFFER_U32@1)

Address: 4412

```
property stream_out1_enable: int
```

When STREAM_OUT#_ENABLE is enabled, the stream out target is generally updated by one value from the stream out buffer per stream scan. For example, there will generally be one instance of e.g. STREAM_OUT0 in the stream scan list, which will cause one STREAM_OUT0_BUFFER value to be consumed and written to STREAM_OUT0_TARGET for every stream scan. The stream scan list could also contain two instances of STREAM_OUT0, in which case two values from STREAM_OUT0_BUFFER value would be consumed and written for every stream scan.

Register: STREAM_OUT1_ENABLE (STREAM_OUT#(0:3)_ENABLE@1)

Address: 4092

```
property stream_out1_loop_num_values: int
```

The number of values, from the end of the array, that will be repeated after reaching the end of supplied data array.

Register: STREAM_OUT1_LOOP_NUM_VALUES (STREAM_OUT#(0:3)_LOOP_NUM_VALUES@1)

Address: 4062

```
property stream_out1_set_loop: int
```

Controls when new data and loop size are used. 1=Use new data immediately. 2=Wait for synch. New data will not be used until a different stream-out channel is set to Synch. 3=Synch. This stream-out# as well as any stream-outs set to synch will start using new data immediately.

Register: STREAM_OUT1_SET_LOOP (STREAM_OUT#(0:3)_SET_LOOP@1)

Address: 4072

```
property stream_out1_target: int
```

Channel that data will be written to. Before writing data to _BUFFER_###, you must write to _TARGET so the device knows how to interpret and store values.

Register: STREAM_OUT1_TARGET (STREAM_OUT#(0:3)_TARGET@1)

Address: 4042

```
property stream_out2: int
```

127) to trigger stream-out updates. When added to the scan list these do count against the max scan rate just like normal input addresses, but they do not return any data in the stream read.

Register: STREAM_OUT2 (STREAM_OUT#(0:3)@2)

Address: 4802

Type:
Include one or more of these registers in STREAM_SCANLIST_ADDRESS#(0

```
property stream_out2_buffer_allocate_num_bytes: int
```

Size of the buffer in bytes as a power of 2. Should be at least twice the size of updates that will be written and no less than 32. The usable buffer size will be equal to the value of this register minus 2 bytes (one 16-bit sample). Before writing data to _BUFFER_###, you must write to _BUFFER_ALLOCATE_NUM_BYTES to allocate RAM for the data. Max is 16384.

Register: STREAM_OUT2_BUFFER_ALLOCATE_NUM_BYTES (STREAM_OUT#(0:3)_BUFFER_ALLOCATE_NUM_BYTES@2)

Address: 4054

```
property stream_out2_buffer_f32: float
```

Data destination when sending floating point data. Appropriate cal constants are used to convert F32 values to 16-bit binary data, and thus each of these values uses 2 bytes of the stream-out buffer. This register is a buffer.

Register: STREAM_OUT2_BUFFER_F32 (STREAM_OUT#(0:3)_BUFFER_F32@2)

Address: 4404

```
property stream_out2_buffer_status: int
```

The number of values in the buffer that are not currently being used.

Register: STREAM_OUT2_BUFFER_STATUS (STREAM_OUT#(0:3)_BUFFER_STATUS@2)

Address: 4084

```
property stream_out2_buffer_u16: int
```

Data destination when sending 16-bit integer data. Each value uses 2 bytes of the stream-out buffer. This register is a buffer.

Register: STREAM_OUT2_BUFFER_U16 (STREAM_OUT#(0:3)_BUFFER_U16@2)

Address: 4422

```
property stream_out2_buffer_u32: int
```

Not used at this time. There are no U32 registers supported in stream-out. This register is a buffer.

Register: STREAM_OUT2_BUFFER_U32 (STREAM_OUT#(0:3)_BUFFER_U32@2)

Address: 4414

```
property stream_out2_enable: int
```

When STREAM_OUT#_ENABLE is enabled, the stream out target is generally updated by one value from the stream out buffer per stream scan. For example, there will generally be one instance of e.g. STREAM_OUT0 in the stream scan list, which will cause one STREAM_OUT0_BUFFER value to be consumed and written to STREAM_OUT0_TARGET for every stream scan. The stream scan list could also contain two instances of STREAM_OUT0, in which case two values from STREAM_OUT0_BUFFER value would be consumed and written for every stream scan.

Register: STREAM_OUT2_ENABLE (STREAM_OUT#(0:3)_ENABLE@2)

Address: 4094

```
property stream_out2_loop_num_values: int
```

The number of values, from the end of the array, that will be repeated after reaching the end of supplied data array.

Register: STREAM_OUT2_LOOP_NUM_VALUES (STREAM_OUT#(0:3)_LOOP_NUM_VALUES@2)

Address: 4064

```
property stream_out2_set_loop: int
```

Controls when new data and loop size are used. 1=Use new data immediately. 2=Wait for synch. New data will not be used until a different stream-out channel is set to Synch. 3=Synch. This stream-out# as well as any stream-outs set to synch will start using new data immediately.

Register: STREAM_OUT2_SET_LOOP (STREAM_OUT#(0:3)_SET_LOOP@2)

Address: 4074

```
property stream_out2_target: int
```

Channel that data will be written to. Before writing data to _BUFFER_###, you must write to _TARGET so the device knows how to interpret and store values.

Register: STREAM_OUT2_TARGET (STREAM_OUT#(0:3)_TARGET@2)

Address: 4044

```
property stream_out3: int
```

127) to trigger stream-out updates. When added to the scan list these do count against the max scan rate just like normal input addresses, but they do not return any data in the stream read.

Register: STREAM_OUT3 (STREAM_OUT#(0:3)@3)

Address: 4803

Type:
Include one or more of these registers in STREAM_SCANLIST_ADDRESS#(0

```
property stream_out3_buffer_allocate_num_bytes: int
```

Size of the buffer in bytes as a power of 2. Should be at least twice the size of updates that will be written and no less than 32. The usable buffer size will be equal to the value of this register minus 2 bytes (one 16-bit sample). Before writing data to _BUFFER_###, you must write to _BUFFER_ALLOCATE_NUM_BYTES to allocate RAM for the data. Max is 16384.

Register: STREAM_OUT3_BUFFER_ALLOCATE_NUM_BYTES (STREAM_OUT#(0:3)_BUFFER_ALLOCATE_NUM_BYTES@3)

Address: 4056

```
property stream_out3_buffer_f32: float
```

Data destination when sending floating point data. Appropriate cal constants are used to convert F32 values to 16-bit binary data, and thus each of these values uses 2 bytes of the stream-out buffer. This register is a buffer.

Register: STREAM_OUT3_BUFFER_F32 (STREAM_OUT#(0:3)_BUFFER_F32@3)

Address: 4406

```
property stream_out3_buffer_status: int
```

The number of values in the buffer that are not currently being used.

Register: STREAM_OUT3_BUFFER_STATUS (STREAM_OUT#(0:3)_BUFFER_STATUS@3)

Address: 4086

```
property stream_out3_buffer_u16: int
```

Data destination when sending 16-bit integer data. Each value uses 2 bytes of the stream-out buffer. This register is a buffer.

Register: STREAM_OUT3_BUFFER_U16 (STREAM_OUT#(0:3)_BUFFER_U16@3)

Address: 4423

```
property stream_out3_buffer_u32: int
```

Not used at this time. There are no U32 registers supported in stream-out. This register is a buffer.

Register: STREAM_OUT3_BUFFER_U32 (STREAM_OUT#(0:3)_BUFFER_U32@3)

Address: 4416

```
property stream_out3_enable: int
```

When STREAM_OUT#_ENABLE is enabled, the stream out target is generally updated by one value from the stream out buffer per stream scan. For example, there will generally be one instance of e.g. STREAM_OUT0 in the stream scan list, which will cause one

STREAM_OUT0_BUFFER value to be consumed and written to STREAM_OUT0_TARGET for every stream scan. The stream scan list could also contain two instances of STREAM_OUT0, in which case two values from STREAM_OUT0_BUFFER value would be consumed and written for every stream scan.

Register: STREAM_OUT3_ENABLE (STREAM_OUT#(0:3)_ENABLE@3)

Address: 4096

```
property stream_out3_loop_num_values: int
```

The number of values, from the end of the array, that will be repeated after reaching the end of supplied data array.

Register: STREAM_OUT3_LOOP_NUM_VALUES (STREAM_OUT#(0:3)_LOOP_NUM_VALUES@3)

Address: 4066

```
property stream_out3_set_loop: int
```

Controls when new data and loop size are used. 1=Use new data immediately. 2=Wait for synch. New data will not be used until a different stream-out channel is set to Synch. 3=Synch. This stream-out# as well as any stream-outs set to synch will start using new data immediately.

Register: STREAM_OUT3_SET_LOOP (STREAM_OUT#(0:3)_SET_LOOP@3)

Address: 4076

```
property stream_out3_target: int
```

Channel that data will be written to. Before writing data to _BUFFER_###, you must write to _TARGET so the device knows how to interpret and store values.

Register: STREAM_OUT3_TARGET (STREAM_OUT#(0:3)_TARGET@3)

Address: 4046

```
property stream_resolution_index: int
```

The resolution index for stream readings. A larger resolution index generally results in lower noise and longer sample times.

Register: STREAM_RESOLUTION_INDEX

Address: 4010

```
property stream_samples_per_packet: int
```

Specifies the number of data points to be sent in the data packet. Only applies to spontaneous mode.

Register: STREAM_SAMPLES_PER_PACKET

Address: 4006

```
property stream_scanlist_address0: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS0 (STREAM_SCANLIST_ADDRESS#(0:127)@0)

Address: 4100

```
property stream_scanlist_address1: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS1 (STREAM_SCANLIST_ADDRESS#(0:127)@1)

Address: 4102

```
property stream_scanlist_address10: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS10 (STREAM_SCANLIST_ADDRESS#(0:127)@10)

Address: 4120

```
property stream_scanlist_address100: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS100 (STREAM_SCANLIST_ADDRESS#(0:127)@100)

Address: 4300

```
property stream_scanlist_address101: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS101 (STREAM_SCANLIST_ADDRESS#(0:127)@101)

Address: 4302

```
property stream_scanlist_address102: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS102 (STREAM_SCANLIST_ADDRESS#(0:127)@102)

Address: 4304

```
property stream_scanlist_address103: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS103 (STREAM_SCANLIST_ADDRESS#(0:127)@103)

Address: 4306

```
property stream_scanlist_address104: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS104 (STREAM_SCANLIST_ADDRESS#(0:127)@104)

Address: 4308

```
property stream_scanlist_address105: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS105 (STREAM_SCANLIST_ADDRESS#(0:127)@105)

Address: 4310

```
property stream_scanlist_address106: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS106 (STREAM_SCANLIST_ADDRESS#(0:127)@106)

Address: 4312

```
property stream_scanlist_address107: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS107 (STREAM_SCANLIST_ADDRESS#(0:127)@107)

Address: 4314

```
property stream_scanlist_address108: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS108 (STREAM_SCANLIST_ADDRESS#(0:127)@108)

Address: 4316

```
property stream_scanlist_address109: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS109 (STREAM_SCANLIST_ADDRESS#(0:127)@109)

Address: 4318

```
property stream_scanlist_address11: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS11 (STREAM_SCANLIST_ADDRESS#(0:127)@11)

Address: 4122

```
property stream_scanlist_address110: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS110 (STREAM_SCANLIST_ADDRESS#(0:127)@110)

Address: 4320

```
property stream_scanlist_address111: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS111 (STREAM_SCANLIST_ADDRESS#(0:127)@111)

Address: 4322

```
property stream_scanlist_address112: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS112 (STREAM_SCANLIST_ADDRESS#(0:127)@112)

Address: 4324

```
property stream_scanlist_address113: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS113 (STREAM_SCANLIST_ADDRESS#(0:127)@113)

Address: 4326

```
property stream_scanlist_address114: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS114 (STREAM_SCANLIST_ADDRESS#(0:127)@114)

Address: 4328

```
property stream_scanlist_address115: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS115 (STREAM_SCANLIST_ADDRESS#(0:127)@115)

Address: 4330

```
property stream_scanlist_address116: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS116 (STREAM_SCANLIST_ADDRESS#(0:127)@116)

Address: 4332

```
property stream_scanlist_address117: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS117 (STREAM_SCANLIST_ADDRESS#(0:127)@117)

Address: 4334

```
property stream_scanlist_address118: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS118 (STREAM_SCANLIST_ADDRESS#(0:127)@118)

Address: 4336

```
property stream_scanlist_address119: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS119 (STREAM_SCANLIST_ADDRESS#(0:127)@119)

Address: 4338

```
property stream_scanlist_address12: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS12 (STREAM_SCANLIST_ADDRESS#(0:127)@12)

Address: 4124

```
property stream_scanlist_address120: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS120 (STREAM_SCANLIST_ADDRESS#(0:127)@120)

Address: 4340

```
property stream_scanlist_address121: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS121 (STREAM_SCANLIST_ADDRESS#(0:127)@121)

Address: 4342

```
property stream_scanlist_address122: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS122 (STREAM_SCANLIST_ADDRESS#(0:127)@122)

Address: 4344

```
property stream_scanlist_address123: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS123 (STREAM_SCANLIST_ADDRESS#(0:127)@123)

Address: 4346

```
property stream_scanlist_address124: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS124 (STREAM_SCANLIST_ADDRESS#(0:127)@124)

Address: 4348

```
property stream_scanlist_address125: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS125 (STREAM_SCANLIST_ADDRESS#(0:127)@125)

Address: 4350

```
property stream_scanlist_address126: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS126 (STREAM_SCANLIST_ADDRESS#(0:127)@126)

Address: 4352

```
property stream_scanlist_address127: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS127 (STREAM_SCANLIST_ADDRESS#(0:127)@127)

Address: 4354

```
property stream_scanlist_address13: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS13 (STREAM_SCANLIST_ADDRESS#(0:127)@13)

Address: 4126

```
property stream_scanlist_address14: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS14 (STREAM_SCANLIST_ADDRESS#(0:127)@14)

Address: 4128

```
property stream_scanlist_address15: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS15 (STREAM_SCANLIST_ADDRESS#(0:127)@15)

Address: 4130

```
property stream_scanlist_address16: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS16 (STREAM_SCANLIST_ADDRESS#(0:127)@16)

Address: 4132

```
property stream_scanlist_address17: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS17 (STREAM_SCANLIST_ADDRESS#(0:127)@17)

Address: 4134

```
property stream_scanlist_address18: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS18 (STREAM_SCANLIST_ADDRESS#(0:127)@18)

Address: 4136

```
property stream_scanlist_address19: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS19 (STREAM_SCANLIST_ADDRESS#(0:127)@19)

Address: 4138

```
property stream_scanlist_address2: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS2 (STREAM_SCANLIST_ADDRESS#(0:127)@2)

Address: 4104

```
property stream_scanlist_address20: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS20 (STREAM_SCANLIST_ADDRESS#(0:127)@20)

Address: 4140

```
property stream_scanlist_address21: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS21 (STREAM_SCANLIST_ADDRESS#(0:127)@21)

Address: 4142

```
property stream_scanlist_address22: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS22 (STREAM_SCANLIST_ADDRESS#(0:127)@22)

Address: 4144

```
property stream_scanlist_address23: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS23 (STREAM_SCANLIST_ADDRESS#(0:127)@23)

Address: 4146

```
property stream_scanlist_address24: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS24 (STREAM_SCANLIST_ADDRESS#(0:127)@24)

Address: 4148

```
property stream_scanlist_address25: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS25 (STREAM_SCANLIST_ADDRESS#(0:127)@25)

Address: 4150

```
property stream_scanlist_address26: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS26 (STREAM_SCANLIST_ADDRESS#(0:127)@26)

Address: 4152

```
property stream_scanlist_address27: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS27 (STREAM_SCANLIST_ADDRESS#(0:127)@27)

Address: 4154

```
property stream_scanlist_address28: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS28 (STREAM_SCANLIST_ADDRESS#(0:127)@28)

Address: 4156

```
property stream_scanlist_address29: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS29 (STREAM_SCANLIST_ADDRESS#(0:127)@29)

Address: 4158

```
property stream_scanlist_address3: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS3 (STREAM_SCANLIST_ADDRESS#(0:127)@3)

Address: 4106

```
property stream_scanlist_address30: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS30 (STREAM_SCANLIST_ADDRESS#(0:127)@30)

Address: 4160

```
property stream_scanlist_address31: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS31 (STREAM_SCANLIST_ADDRESS#(0:127)@31)

Address: 4162

```
property stream_scanlist_address32: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS32 (STREAM_SCANLIST_ADDRESS#(0:127)@32)

Address: 4164

```
property stream_scanlist_address33: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS33 (STREAM_SCANLIST_ADDRESS#(0:127)@33)

Address: 4166

```
property stream_scanlist_address34: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS34 (STREAM_SCANLIST_ADDRESS#(0:127)@34)

Address: 4168

```
property stream_scanlist_address35: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS35 (STREAM_SCANLIST_ADDRESS#(0:127)@35)

Address: 4170

```
property stream_scanlist_address36: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS36 (STREAM_SCANLIST_ADDRESS#(0:127)@36)

Address: 4172

```
property stream_scanlist_address37: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS37 (STREAM_SCANLIST_ADDRESS#(0:127)@37)

Address: 4174

```
property stream_scanlist_address38: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS38 (STREAM_SCANLIST_ADDRESS#(0:127)@38)

Address: 4176

```
property stream_scanlist_address39: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS39 (STREAM_SCANLIST_ADDRESS#(0:127)@39)

Address: 4178

```
property stream_scanlist_address4: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS4 (STREAM_SCANLIST_ADDRESS#(0:127)@4)

Address: 4108

```
property stream_scanlist_address40: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS40 (STREAM_SCANLIST_ADDRESS#(0:127)@40)

Address: 4180

```
property stream_scanlist_address41: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS41 (STREAM_SCANLIST_ADDRESS#(0:127)@41)

Address: 4182

```
property stream_scanlist_address42: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS42 (STREAM_SCANLIST_ADDRESS#(0:127)@42)

Address: 4184

```
property stream_scanlist_address43: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS43 (STREAM_SCANLIST_ADDRESS#(0:127)@43)

Address: 4186

```
property stream_scanlist_address44: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS44 (STREAM_SCANLIST_ADDRESS#(0:127)@44)

Address: 4188

```
property stream_scanlist_address45: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS45 (STREAM_SCANLIST_ADDRESS#(0:127)@45)

Address: 4190

```
property stream_scanlist_address46: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS46 (STREAM_SCANLIST_ADDRESS#(0:127)@46)

Address: 4192

```
property stream_scanlist_address47: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS47 (STREAM_SCANLIST_ADDRESS#(0:127)@47)

Address: 4194

```
property stream_scanlist_address48: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS48 (STREAM_SCANLIST_ADDRESS#(0:127)@48)

Address: 4196

```
property stream_scanlist_address49: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS49 (STREAM_SCANLIST_ADDRESS#(0:127)@49)

Address: 4198

```
property stream_scanlist_address5: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS5 (STREAM_SCANLIST_ADDRESS#(0:127)@5)

Address: 4110

```
property stream_scanlist_address50: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS50 (STREAM_SCANLIST_ADDRESS#(0:127)@50)

Address: 4200

```
property stream_scanlist_address51: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS51 (STREAM_SCANLIST_ADDRESS#(0:127)@51)

Address: 4202

```
property stream_scanlist_address52: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS52 (STREAM_SCANLIST_ADDRESS#(0:127)@52)

Address: 4204

```
property stream_scanlist_address53: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS53 (STREAM_SCANLIST_ADDRESS#(0:127)@53)

Address: 4206

```
property stream_scanlist_address54: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS54 (STREAM_SCANLIST_ADDRESS#(0:127)@54)

Address: 4208

```
property stream_scanlist_address55: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS55 (STREAM_SCANLIST_ADDRESS#(0:127)@55)

Address: 4210

```
property stream_scanlist_address56: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS56 (STREAM_SCANLIST_ADDRESS#(0:127)@56)

Address: 4212

```
property stream_scanlist_address57: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS57 (STREAM_SCANLIST_ADDRESS#(0:127)@57)

Address: 4214

```
property stream_scanlist_address58: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS58 (STREAM_SCANLIST_ADDRESS#(0:127)@58)

Address: 4216

```
property stream_scanlist_address59: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS59 (STREAM_SCANLIST_ADDRESS#(0:127)@59)

Address: 4218

```
property stream_scanlist_address6: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS6 (STREAM_SCANLIST_ADDRESS#(0:127)@6)

Address: 4112

```
property stream_scanlist_address60: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS60 (STREAM_SCANLIST_ADDRESS#(0:127)@60)

Address: 4220

```
property stream_scanlist_address61: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS61 (STREAM_SCANLIST_ADDRESS#(0:127)@61)

Address: 4222

```
property stream_scanlist_address62: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS62 (STREAM_SCANLIST_ADDRESS#(0:127)@62)

Address: 4224

```
property stream_scanlist_address63: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS63 (STREAM_SCANLIST_ADDRESS#(0:127)@63)

Address: 4226

```
property stream_scanlist_address64: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS64 (STREAM_SCANLIST_ADDRESS#(0:127)@64)

Address: 4228

```
property stream_scanlist_address65: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS65 (STREAM_SCANLIST_ADDRESS#(0:127)@65)

Address: 4230

```
property stream_scanlist_address66: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS66 (STREAM_SCANLIST_ADDRESS#(0:127)@66)

Address: 4232

```
property stream_scanlist_address67: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS67 (STREAM_SCANLIST_ADDRESS#(0:127)@67)

Address: 4234

```
property stream_scanlist_address68: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS68 (STREAM_SCANLIST_ADDRESS#(0:127)@68)

Address: 4236

```
property stream_scanlist_address69: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS69 (STREAM_SCANLIST_ADDRESS#(0:127)@69)

Address: 4238

```
property stream_scanlist_address7: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS7 (STREAM_SCANLIST_ADDRESS#(0:127)@7)

Address: 4114

```
property stream_scanlist_address70: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS70 (STREAM_SCANLIST_ADDRESS#(0:127)@70)

Address: 4240

```
property stream_scanlist_address71: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS71 (STREAM_SCANLIST_ADDRESS#(0:127)@71)

Address: 4242

```
property stream_scanlist_address72: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS72 (STREAM_SCANLIST_ADDRESS#(0:127)@72)

Address: 4244

```
property stream_scanlist_address73: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS73 (STREAM_SCANLIST_ADDRESS#(0:127)@73)

Address: 4246

```
property stream_scanlist_address74: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS74 (STREAM_SCANLIST_ADDRESS#(0:127)@74)

Address: 4248

```
property stream_scanlist_address75: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS75 (STREAM_SCANLIST_ADDRESS#(0:127)@75)

Address: 4250

```
property stream_scanlist_address76: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS76 (STREAM_SCANLIST_ADDRESS#(0:127)@76)

Address: 4252

```
property stream_scanlist_address77: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS77 (STREAM_SCANLIST_ADDRESS#(0:127)@77)

Address: 4254

```
property stream_scanlist_address78: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS78 (STREAM_SCANLIST_ADDRESS#(0:127)@78)

Address: 4256

```
property stream_scanlist_address79: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS79 (STREAM_SCANLIST_ADDRESS#(0:127)@79)

Address: 4258

```
property stream_scanlist_address8: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS8 (STREAM_SCANLIST_ADDRESS#(0:127)@8)

Address: 4116

```
property stream_scanlist_address80: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS80 (STREAM_SCANLIST_ADDRESS#(0:127)@80)

Address: 4260

```
property stream_scanlist_address81: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS81 (STREAM_SCANLIST_ADDRESS#(0:127)@81)

Address: 4262

```
property stream_scanlist_address82: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS82 (STREAM_SCANLIST_ADDRESS#(0:127)@82)

Address: 4264

```
property stream_scanlist_address83: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS83 (STREAM_SCANLIST_ADDRESS#(0:127)@83)

Address: 4266

```
property stream_scanlist_address84: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS84 (STREAM_SCANLIST_ADDRESS#(0:127)@84)

Address: 4268

```
property stream_scanlist_address85: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS85 (STREAM_SCANLIST_ADDRESS#(0:127)@85)

Address: 4270

```
property stream_scanlist_address86: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS86 (STREAM_SCANLIST_ADDRESS#(0:127)@86)

Address: 4272

```
property stream_scanlist_address87: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS87 (STREAM_SCANLIST_ADDRESS#(0:127)@87)

Address: 4274

```
property stream_scanlist_address88: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS88 (STREAM_SCANLIST_ADDRESS#(0:127)@88)

Address: 4276

```
property stream_scanlist_address89: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS89 (STREAM_SCANLIST_ADDRESS#(0:127)@89)

Address: 4278

```
property stream_scanlist_address9: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS9 (STREAM_SCANLIST_ADDRESS#(0:127)@9)

Address: 4118

```
property stream_scanlist_address90: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS90 (STREAM_SCANLIST_ADDRESS#(0:127)@90)

Address: 4280

```
property stream_scanlist_address91: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS91 (STREAM_SCANLIST_ADDRESS#(0:127)@91)

Address: 4282

```
property stream_scanlist_address92: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS92 (STREAM_SCANLIST_ADDRESS#(0:127)@92)

Address: 4284

```
property stream_scanlist_address93: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS93 (STREAM_SCANLIST_ADDRESS#(0:127)@93)

Address: 4286

```
property stream_scanlist_address94: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS94 (STREAM_SCANLIST_ADDRESS#(0:127)@94)

Address: 4288

```
property stream_scanlist_address95: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS95 (STREAM_SCANLIST_ADDRESS#(0:127)@95)

Address: 4290

```
property stream_scanlist_address96: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS96 (STREAM_SCANLIST_ADDRESS#(0:127)@96)

Address: 4292

```
property stream_scanlist_address97: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS97 (STREAM_SCANLIST_ADDRESS#(0:127)@97)

Address: 4294

```
property stream_scanlist_address98: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS98 (STREAM_SCANLIST_ADDRESS#(0:127)@98)

Address: 4296

```
property stream_scanlist_address99: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS99 (STREAM_SCANLIST_ADDRESS#(0:127)@99)

Address: 4298

```
property stream_scanrate_hz: float
```

Write a value to specify the number of times per second that all channels in the stream scanlist will be read. Max stream speeds are based on Sample Rate which is NumChannels*ScanRate. Has no effect when using an external clock. A read of this register returns the actual scan rate, which can be slightly different due to rounding. For scan rates >152.588, the actual scan interval is multiples of 100 ns. Assuming a core clock of 80 MHz the

internal roll value is (80M/(8*DesiredScanRate))-1 and the actual scan rate is then 80M/(8*(RollValue+1). For slower scan rates the scan interval resolution is changed to 1 us, 10 us, 100 us, or 1 ms as needed to achieve the longer intervals.

Register: STREAM_SCANRATE_HZ

Address: 4002

```
property stream_settling_us: float
```

Time in microseconds to allow signals to settle after switching the mux. Does not apply to the 1st channel in the scan list, as that settling is controlled by scan rate (the time from the last channel until the start of the next scan). Default=0. When set to less than 1, automatic settling will be used. The automatic settling behavior varies by device.

Register: STREAM_SETTLING_US

Address: 4008

```
property stream_start_time_stamp: int
```

This register stores the value of CORE_TIMER at the start of the first stream scan. Note that the first stream scan happens 1 scan period after STREAM_ENABLE is set to 1.

Register: STREAM_START_TIME_STAMP

Address: 4026

```
property system_counter_10khz: int
```

A 10 kHz counter synchronized to RTC_TIME_S. This register can be appended to RTC_TIME_S as the decimal portion to get 0.1 ms resolution.

Register: SYSTEM_COUNTER_10KHZ

Address: 61502

```
property system_reboot: int
```

Issues a device reboot. Must write 0x4C4Axxxx, where xxxx is number of 50ms ticks to wait before reboot. To reboot immediately write 0x4C4A0000 (d1279918080).

Register: SYSTEM_REBOOT

Address: 61998

```
property system_timer_20hz: int
```

Internal 32-bit system timer running at 20Hz.

Register: SYSTEM_TIMER_20HZ

Address: 61522

```
property tdac0: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC0 (TDAC#(0:21)@0)

Address: 30000

```
property tdac1: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC1 (TDAC#(0:21)@1)

Address: 30002

```
property tdac10: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC10 (TDAC#(0:21)@10)

Address: 30020

```
property tdac11: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC11 (TDAC#(0:21)@11)

Address: 30022

```
property tdac12: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC12 (TDAC#(0:21)@12)

Address: 30024

```
property tdac13: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC13 (TDAC#(0:21)@13)

Address: 30026

```
property tdac14: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC14 (TDAC#(0:21)@14)

Address: 30028

```
property tdac15: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC15 (TDAC#(0:21)@15)

Address: 30030

```
property tdac16: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC16 (TDAC#(0:21)@16)

Address: 30032

```
property tdac17: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC17 (TDAC#(0:21)@17)

Address: 30034

```
property tdac18: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC18 (TDAC#(0:21)@18)

Address: 30036

```
property tdac19: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC19 (TDAC#(0:21)@19)

Address: 30038

```
property tdac2: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC2 (TDAC#(0:21)@2)

Address: 30004

```
property tdac20: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC20 (TDAC#(0:21)@20)

Address: 30040

```
property tdac21: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC21 (TDAC#(0:21)@21)

Address: 30042

```
property tdac3: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC3 (TDAC#(0:21)@3)

Address: 30006

```
property tdac4: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC4 (TDAC#(0:21)@4)

Address: 30008

```
property tdac5: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC5 (TDAC#(0:21)@5)

Address: 30010

```
property tdac6: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC6 (TDAC#(0:21)@6)

Address: 30012

```
property tdac7: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC7 (TDAC#(0:21)@7)

Address: 30014

```
property tdac8: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC8 (TDAC#(0:21)@8)

Address: 30016

```
property tdac9: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC9 (TDAC#(0:21)@9)

Address: 30018

```
property tdac_serial_number: int
```

Returns the serial number of an LJTick-DAC, and forces a re-read of the calibration constants. Which LJTDAC is determined by the last write to TDAC# ... whether it was successful or not.

Register: TDAC_SERIAL_NUMBER

Address: 55200

```
property tdac_speed_throttle: int
```

Sets the I2C clock speed that will be used when communicating with the TDAC. Default value is 65516. See I2C_SPEED_THROTTLE for more detail.

Register: TDAC_SPEED_THROTTLE

Address: 55202

```
property temperature_air_k: float
```

Returns the estimated ambient air temperature just outside of the device in its red plastic enclosure. This register is equal to TEMPERATURE_DEVICE_K - 4.3. If Ethernet and/or WiFi is enabled, subtract an extra 0.6 for each.

Register: TEMPERATURE_AIR_K

Address: 60050

```
property temperature_device_k: float
```

Takes a reading from the internal temperature sensor using range=+/-10V and resolution=8, and applies the formula Volts*-92.6+467.6 to return kelvins.

Register: TEMPERATURE_DEVICE_K

Address: 60052

```
property test: int
```

A read of this test register should always return 0x00112233 or d1122867. If your software has the word swap quirk, you will incorrectly read 0x22330011 or 573767697. If your software has the address-1 quirk, a UINT16 (1-register) read from 55101 will incorrectly return 0x0011 (should read 0x2233). This register is unlike others, in that it allows you can read a single word from 55100 or 55101, or of course 2 words from 55100.

Register: TEST

Address: 55100

```
property test_float32: float
```

Write a value and read back to test FLOAT32 operation. Default is 0xC61C3C00 or -9999.0. If your software has the word swap quirk, the default will incorrectly read 0x3C00C61C or 0.00786.

Register: TEST_FLOAT32

Address: 55124

```
property test_int32: int
```

Write a value and read back to test INT32 operation. Default is 0x8899AABB or d-2003195205. If your software has the word swap quirk, the default will incorrectly read 0xAABB8899 or -1430550375.

Register: TEST_INT32

Address: 55122

```
property test_uint16: int
```

Write a value and read back to test UINT16 operation. Default is 0x0011 or d17.

Register: TEST_UINT16

Address: 55110

```
property test_uint32: int
```

Write a value and read back to test UINT32 operation. Default is 0x00112233 or d1122867. If your software has the word swap quirk, the default will incorrectly read 0x22330011 or 573767697.

Register: TEST_UINT32

Address: 55120

```
property user_ram0_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM0_F32 (USER_RAM#(0:39)_F32@0)

Address: 46000

```
property user_ram0_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM0_I32 (USER_RAM#(0:9)_I32@0)

Address: 46080

```
property user_ram0_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM0_U16 (USER_RAM#(0:19)_U16@0)

Address: 46180

```
property user_ram0_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM0_U32 (USER_RAM#(0:39)_U32@0)

Address: 46100

```
property user_ram10_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM10_F32 (USER_RAM#(0:39)_F32@10)

Address: 46020

```
property user_ram10_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM10_U16 (USER_RAM#(0:19)_U16@10)

Address: 46190

```
property user_ram10_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM10_U32 (USER_RAM#(0:39)_U32@10)

Address: 46120

```
property user_ram11_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM11_F32 (USER_RAM#(0:39)_F32@11)

Address: 46022

```
property user_ram11_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM11_U16 (USER_RAM#(0:19)_U16@11)

Address: 46191

```
property user_ram11_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM11_U32 (USER_RAM#(0:39)_U32@11)

Address: 46122

```
property user_ram12_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM12_F32 (USER_RAM#(0:39)_F32@12)

Address: 46024

```
property user_ram12_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM12_U16 (USER_RAM#(0:19)_U16@12)

Address: 46192

```
property user_ram12_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM12_U32 (USER_RAM#(0:39)_U32@12)

Address: 46124

```
property user_ram13_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM13_F32 (USER_RAM#(0:39)_F32@13)

Address: 46026

```
property user_ram13_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM13_U16 (USER_RAM#(0:19)_U16@13)

Address: 46193

```
property user_ram13_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM13_U32 (USER_RAM#(0:39)_U32@13)

Address: 46126

```
property user_ram14_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM14_F32 (USER_RAM#(0:39)_F32@14)

Address: 46028

```
property user_ram14_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM14_U16 (USER_RAM#(0:19)_U16@14)

Address: 46194

```
property user_ram14_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM14_U32 (USER_RAM#(0:39)_U32@14)

Address: 46128

```
property user_ram15_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM15_F32 (USER_RAM#(0:39)_F32@15)

Address: 46030

```
property user_ram15_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM15_U16 (USER_RAM#(0:19)_U16@15)

Address: 46195

```
property user_ram15_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM15_U32 (USER_RAM#(0:39)_U32@15)

Address: 46130

```
property user_ram16_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM16_F32 (USER_RAM#(0:39)_F32@16)

Address: 46032

```
property user_ram16_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM16_U16 (USER_RAM#(0:19)_U16@16)

Address: 46196

```
property user_ram16_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM16_U32 (USER_RAM#(0:39)_U32@16)

Address: 46132

```
property user_ram17_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM17_F32 (USER_RAM#(0:39)_F32@17)

Address: 46034

```
property user_ram17_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM17_U16 (USER_RAM#(0:19)_U16@17)

Address: 46197

```
property user_ram17_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM17_U32 (USER_RAM#(0:39)_U32@17)

Address: 46134

```
property user_ram18_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM18_F32 (USER_RAM#(0:39)_F32@18)

Address: 46036

```
property user_ram18_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM18_U16 (USER_RAM#(0:19)_U16@18)

Address: 46198

```
property user_ram18_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM18_U32 (USER_RAM#(0:39)_U32@18)

Address: 46136

```
property user_ram19_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM19_F32 (USER_RAM#(0:39)_F32@19)

Address: 46038

```
property user_ram19_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM19_U16 (USER_RAM#(0:19)_U16@19)

Address: 46199

```
property user_ram19_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM19_U32 (USER_RAM#(0:39)_U32@19)

Address: 46138

```
property user_ram1_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM1_F32 (USER_RAM#(0:39)_F32@1)

Address: 46002

```
property user_ram1_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM1_I32 (USER_RAM#(0:9)_I32@1)

Address: 46082

```
property user_ram1_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM1_U16 (USER_RAM#(0:19)_U16@1)

Address: 46181

```
property user_ram1_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM1_U32 (USER_RAM#(0:39)_U32@1)

Address: 46102

```
property user_ram20_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM20_F32 (USER_RAM#(0:39)_F32@20)

Address: 46040

```
property user_ram20_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM20_U32 (USER_RAM#(0:39)_U32@20)

Address: 46140

```
property user_ram21_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM21_F32 (USER_RAM#(0:39)_F32@21)

Address: 46042

```
property user_ram21_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM21_U32 (USER_RAM#(0:39)_U32@21)

Address: 46142

```
property user_ram22_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM22_F32 (USER_RAM#(0:39)_F32@22)

Address: 46044

```
property user_ram22_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM22_U32 (USER_RAM#(0:39)_U32@22)

Address: 46144

```
property user_ram23_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM23_F32 (USER_RAM#(0:39)_F32@23)

Address: 46046

```
property user_ram23_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM23_U32 (USER_RAM#(0:39)_U32@23)

Address: 46146

```
property user_ram24_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM24_F32 (USER_RAM#(0:39)_F32@24)

Address: 46048

```
property user_ram24_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM24_U32 (USER_RAM#(0:39)_U32@24)

Address: 46148

```
property user_ram25_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM25_F32 (USER_RAM#(0:39)_F32@25)

Address: 46050

```
property user_ram25_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM25_U32 (USER_RAM#(0:39)_U32@25)

Address: 46150

```
property user_ram26_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM26_F32 (USER_RAM#(0:39)_F32@26)

Address: 46052

```
property user_ram26_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM26_U32 (USER_RAM#(0:39)_U32@26)

Address: 46152

```
property user_ram27_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM27_F32 (USER_RAM#(0:39)_F32@27)

Address: 46054

```
property user_ram27_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM27_U32 (USER_RAM#(0:39)_U32@27)

Address: 46154

```
property user_ram28_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM28_F32 (USER_RAM#(0:39)_F32@28)

Address: 46056

```
property user_ram28_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM28_U32 (USER_RAM#(0:39)_U32@28)

Address: 46156

```
property user_ram29_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM29_F32 (USER_RAM#(0:39)_F32@29)

Address: 46058

```
property user_ram29_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM29_U32 (USER_RAM#(0:39)_U32@29)

Address: 46158

```
property user_ram2_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM2_F32 (USER_RAM#(0:39)_F32@2)

Address: 46004

```
property user_ram2_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM2_I32 (USER_RAM#(0:9)_I32@2)

Address: 46084

```
property user_ram2_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM2_U16 (USER_RAM#(0:19)_U16@2)

Address: 46182

```
property user_ram2_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM2_U32 (USER_RAM#(0:39)_U32@2)

Address: 46104

```
property user_ram30_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM30_F32 (USER_RAM#(0:39)_F32@30)

Address: 46060

```
property user_ram30_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM30_U32 (USER_RAM#(0:39)_U32@30)

Address: 46160

```
property user_ram31_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM31_F32 (USER_RAM#(0:39)_F32@31)

Address: 46062

```
property user_ram31_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM31_U32 (USER_RAM#(0:39)_U32@31)

Address: 46162

```
property user_ram32_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM32_F32 (USER_RAM#(0:39)_F32@32)

Address: 46064

```
property user_ram32_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM32_U32 (USER_RAM#(0:39)_U32@32)

Address: 46164

```
property user_ram33_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM33_F32 (USER_RAM#(0:39)_F32@33)

Address: 46066

```
property user_ram33_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM33_U32 (USER_RAM#(0:39)_U32@33)

Address: 46166

```
property user_ram34_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM34_F32 (USER_RAM#(0:39)_F32@34)

Address: 46068

```
property user_ram34_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM34_U32 (USER_RAM#(0:39)_U32@34)

Address: 46168

```
property user_ram35_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM35_F32 (USER_RAM#(0:39)_F32@35)

Address: 46070

```
property user_ram35_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM35_U32 (USER_RAM#(0:39)_U32@35)

Address: 46170

```
property user_ram36_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM36_F32 (USER_RAM#(0:39)_F32@36)

Address: 46072

```
property user_ram36_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM36_U32 (USER_RAM#(0:39)_U32@36)

Address: 46172

```
property user_ram37_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM37_F32 (USER_RAM#(0:39)_F32@37)

Address: 46074

```
property user_ram37_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM37_U32 (USER_RAM#(0:39)_U32@37)

Address: 46174

```
property user_ram38_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM38_F32 (USER_RAM#(0:39)_F32@38)

Address: 46076

```
property user_ram38_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM38_U32 (USER_RAM#(0:39)_U32@38)

Address: 46176

```
property user_ram39_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM39_F32 (USER_RAM#(0:39)_F32@39)

Address: 46078

```
property user_ram39_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM39_U32 (USER_RAM#(0:39)_U32@39)

Address: 46178

```
property user_ram3_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM3_F32 (USER_RAM#(0:39)_F32@3)

Address: 46006

```
property user_ram3_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM3_I32 (USER_RAM#(0:9)_I32@3)

Address: 46086

```
property user_ram3_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM3_U16 (USER_RAM#(0:19)_U16@3)

Address: 46183

```
property user_ram3_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM3_U32 (USER_RAM#(0:39)_U32@3)

Address: 46106

```
property user_ram4_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM4_F32 (USER_RAM#(0:39)_F32@4)

Address: 46008

```
property user_ram4_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM4_I32 (USER_RAM#(0:9)_I32@4)

Address: 46088

```
property user_ram4_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM4_U16 (USER_RAM#(0:19)_U16@4)

Address: 46184

```
property user_ram4_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM4_U32 (USER_RAM#(0:39)_U32@4)

Address: 46108

```
property user_ram5_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM5_F32 (USER_RAM#(0:39)_F32@5)

Address: 46010

```
property user_ram5_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM5_I32 (USER_RAM#(0:9)_I32@5)

Address: 46090

```
property user_ram5_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM5_U16 (USER_RAM#(0:19)_U16@5)

Address: 46185

```
property user_ram5_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM5_U32 (USER_RAM#(0:39)_U32@5)

Address: 46110

```
property user_ram6_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM6_F32 (USER_RAM#(0:39)_F32@6)

Address: 46012

```
property user_ram6_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM6_I32 (USER_RAM#(0:9)_I32@6)

Address: 46092

```
property user_ram6_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM6_U16 (USER_RAM#(0:19)_U16@6)

Address: 46186

```
property user_ram6_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM6_U32 (USER_RAM#(0:39)_U32@6)

Address: 46112

```
property user_ram7_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM7_F32 (USER_RAM#(0:39)_F32@7)

Address: 46014

```
property user_ram7_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM7_I32 (USER_RAM#(0:9)_I32@7)

Address: 46094

```
property user_ram7_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM7_U16 (USER_RAM#(0:19)_U16@7)

Address: 46187

```
property user_ram7_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM7_U32 (USER_RAM#(0:39)_U32@7)

Address: 46114

```
property user_ram8_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM8_F32 (USER_RAM#(0:39)_F32@8)

Address: 46016

```
property user_ram8_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM8_I32 (USER_RAM#(0:9)_I32@8)

Address: 46096

```
property user_ram8_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM8_U16 (USER_RAM#(0:19)_U16@8)

Address: 46188

```
property user_ram8_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM8_U32 (USER_RAM#(0:39)_U32@8)

Address: 46116

```
property user_ram9_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM9_F32 (USER_RAM#(0:39)_F32@9)

Address: 46018

```
property user_ram9_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM9_I32 (USER_RAM#(0:9)_I32@9)

Address: 46098

```
property user_ram9_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM9_U16 (USER_RAM#(0:19)_U16@9)

Address: 46189

```
property user_ram9_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM9_U32 (USER_RAM#(0:39)_U32@9)

Address: 46118

```
property user_ram_fifo0_allocate_num_bytes: int
```

Allocate memory for a FIFO buffer. Number of bytes should be sufficient to store users max transfer array size. Note that FLOAT32, INT32, and UINT32 require 4 bytes per value, and UINT16 require 2 bytes per value. Maximum size is limited by available memory. Care should be taken to conserve enough memory for other operations such as AIN_EF, Lua, Stream etc.

Register: USER_RAM_FIFO0_ALLOCATE_NUM_BYTES
(USER_RAM_FIFO#(0:3)_ALLOCATE_NUM_BYTES@0)

Address: 47900

```
property user_ram_fifo0_data_f32: float
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO0_DATA_F32 (USER_RAM_FIFO#(0:3)_DATA_F32@0)

Address: 47030

```
property user_ram_fifo0_data_i32: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO0_DATA_I32 (USER_RAM_FIFO#(0:3)_DATA_I32@0)

Address: 47020

```
property user_ram_fifo0_data_u16: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO0_DATA_U16 (USER_RAM_FIFO#(0:3)_DATA_U16@0)

Address: 47000

```
property user_ram_fifo0_data_u32: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO0_DATA_U32 (USER_RAM_FIFO#(0:3)_DATA_U32@0)

Address: 47010

```
property user_ram_fifo0_empty: int
```

Write any value to this register to efficiently empty, flush, or otherwise clear data from the FIFO.

Register: USER_RAM_FIFO0_EMPTY (USER_RAM_FIFO#(0:3)_EMPTY@0)

Address: 47930

```
property user_ram_fifo0_num_bytes_in_fifo: int
```

Poll this register to see when new data is available/ready. Each read of the FIFO buffer decreases this value, and each write to the FIFO buffer increases this value.

At any point in time, the following equation holds: Nbytes = Nwritten - Nread.

Register: USER_RAM_FIFO0_NUM_BYTES_IN_FIFO
(USER_RAM_FIFO#(0:3)_NUM_BYTES_IN_FIFO@0)

Address: 47910

```
property user_ram_fifo1_allocate_num_bytes: int
```

Allocate memory for a FIFO buffer. Number of bytes should be sufficient to store users max transfer array size. Note that FLOAT32, INT32, and UINT32 require 4 bytes per value, and UINT16 require 2 bytes per value. Maximum size is limited by available memory. Care should be taken to conserve enough memory for other operations such as AIN_EF, Lua, Stream etc.

Register: USER_RAM_FIFO1_ALLOCATE_NUM_BYTES
(USER_RAM_FIFO#(0:3)_ALLOCATE_NUM_BYTES@1)

Address: 47902

```
property user_ram_fifo1_data_f32: float
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO1_DATA_F32 (USER_RAM_FIFO#(0:3)_DATA_F32@1)

Address: 47032

```
property user_ram_fifo1_data_i32: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO1_DATA_I32 (USER_RAM_FIFO#(0:3)_DATA_I32@1)

Address: 47022

```
property user_ram_fifo1_data_u16: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO1_DATA_U16 (USER_RAM_FIFO#(0:3)_DATA_U16@1)

Address: 47001

```
property user_ram_fifo1_data_u32: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO1_DATA_U32 (USER_RAM_FIFO#(0:3)_DATA_U32@1)

Address: 47012

```
property user_ram_fifo1_empty: int
```

Write any value to this register to efficiently empty, flush, or otherwise clear data from the FIFO.

Register: USER_RAM_FIFO1_EMPTY (USER_RAM_FIFO#(0:3)_EMPTY@1)

Address: 47932

```
property user_ram_fifo1_num_bytes_in_fifo: int
```

Poll this register to see when new data is available/ready. Each read of the FIFO buffer decreases this value, and each write to the FIFO buffer increases this value.

At any point in time, the following equation holds: Nbytes = Nwritten - Nread.

Register: USER_RAM_FIFO1_NUM_BYTES_IN_FIFO
(USER_RAM_FIFO#(0:3)_NUM_BYTES_IN_FIFO@1)

Address: 47912

```
property user_ram_fifo2_allocate_num_bytes: int
```

Allocate memory for a FIFO buffer. Number of bytes should be sufficient to store users max transfer array size. Note that FLOAT32, INT32, and UINT32 require 4 bytes per value, and UINT16 require 2 bytes per value. Maximum size is limited by available memory. Care should be taken to conserve enough memory for other operations such as AIN_EF, Lua, Stream etc.

Register: USER_RAM_FIFO2_ALLOCATE_NUM_BYTES
(USER_RAM_FIFO#(0:3)_ALLOCATE_NUM_BYTES@2)

Address: 47904

```
property user_ram_fifo2_data_f32: float
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO2_DATA_F32 (USER_RAM_FIFO#(0:3)_DATA_F32@2)

Address: 47034

```
property user_ram_fifo2_data_i32: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO2_DATA_I32 (USER_RAM_FIFO#(0:3)_DATA_I32@2)

Address: 47024

```
property user_ram_fifo2_data_u16: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO2_DATA_U16 (USER_RAM_FIFO#(0:3)_DATA_U16@2)

Address: 47002

```
property user_ram_fifo2_data_u32: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO2_DATA_U32 (USER_RAM_FIFO#(0:3)_DATA_U32@2)

Address: 47014

```
property user_ram_fifo2_empty: int
```

Write any value to this register to efficiently empty, flush, or otherwise clear data from the FIFO.

Register: USER_RAM_FIFO2_EMPTY (USER_RAM_FIFO#(0:3)_EMPTY@2)

Address: 47934

```
property user_ram_fifo2_num_bytes_in_fifo: int
```

Poll this register to see when new data is available/ready. Each read of the FIFO buffer decreases this value, and each write to the FIFO buffer increases this value.

At any point in time, the following equation holds: Nbytes = Nwritten - Nread.

Register: USER_RAM_FIFO2_NUM_BYTES_IN_FIFO
(USER_RAM_FIFO#(0:3)_NUM_BYTES_IN_FIFO@2)

Address: 47914

```
property user_ram_fifo3_allocate_num_bytes: int
```

Allocate memory for a FIFO buffer. Number of bytes should be sufficient to store users max transfer array size. Note that FLOAT32, INT32, and UINT32 require 4 bytes per value, and UINT16 require 2 bytes per value. Maximum size is limited by available memory. Care should be taken to conserve enough memory for other operations such as AIN_EF, Lua, Stream etc.

Register: USER_RAM_FIFO3_ALLOCATE_NUM_BYTES
(USER_RAM_FIFO#(0:3)_ALLOCATE_NUM_BYTES@3)

Address: 47906

```
property user_ram_fifo3_data_f32: float
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO3_DATA_F32 (USER_RAM_FIFO#(0:3)_DATA_F32@3)

Address: 47036

```
property user_ram_fifo3_data_i32: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO3_DATA_I32 (USER_RAM_FIFO#(0:3)_DATA_I32@3)

Address: 47026

```
property user_ram_fifo3_data_u16: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO3_DATA_U16 (USER_RAM_FIFO#(0:3)_DATA_U16@3)

Address: 47003

```
property user_ram_fifo3_data_u32: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO3_DATA_U32 (USER_RAM_FIFO#(0:3)_DATA_U32@3)

Address: 47016

```
property user_ram_fifo3_empty: int
```

Write any value to this register to efficiently empty, flush, or otherwise clear data from the FIFO.

Register: USER_RAM_FIFO3_EMPTY (USER_RAM_FIFO#(0:3)_EMPTY@3)

Address: 47936

```
property user_ram_fifo3_num_bytes_in_fifo: int
```

Poll this register to see when new data is available/ready. Each read of the FIFO buffer decreases this value, and each write to the FIFO buffer increases this value.

At any point in time, the following equation holds: Nbytes = Nwritten - Nread.

Register: USER_RAM_FIFO3_NUM_BYTES_IN_FIFO
(USER_RAM_FIFO#(0:3)_NUM_BYTES_IN_FIFO@3)

Address: 47916

```
property wait_us_blocking: int
```

Delays for x microseconds. Range is 0-100000. This operation is Blocking. While a blocking function is running no other registers can be read / written.

Register: WAIT_US_BLOCKING

Address: 61590

```
property watchdog_advanced_default: int
```

A single binary-encoded value where each bit is an advanced option. If bit 0 is set, IO_CONFIG_SET_CURRENT_TO_FACTORY will be done on timeout. If bit 1 is set, IO_CONFIG_SET_CURRENT_TO_DEFAULT will be done on timeout.

Register: WATCHDOG_ADVANCED_DEFAULT

Address: 61602

```
property watchdog_dac0_default: float
```

The voltage of DAC0 after a Watchdog timeout.

Register: WATCHDOG_DAC0_DEFAULT

Address: 61642

```
property watchdog_dac0_enable_default: int
```

Timeout action: Set to 1 to enable DAC0 update on watchdog timeout.

Register: WATCHDOG_DAC0_ENABLE_DEFAULT

Address: 61640

```
property watchdog_dac1_default: float
```

The voltage of DAC1 after a Watchdog timeout.

Register: WATCHDOG_DAC1_DEFAULT

Address: 61652

```
property watchdog_dac1_enable_default: int
```

Timeout action: Set to 1 to enable DAC1 update on watchdog timeout.

Register: WATCHDOG_DAC1_ENABLE_DEFAULT

Address: 61650

```
property watchdog_dio_direction_default: int
```

The direction input/output of the digital I/O after a Watchdog timeout. See DIO_DIRECTION.

Register: WATCHDOG_DIO_DIRECTION_DEFAULT

Address: 61634

```
property watchdog_dio_enable_default: int
```

Timeout action: Set to 1 to enable DIO update on watchdog timeout.

Register: WATCHDOG_DIO_ENABLE_DEFAULT

Address: 61630

```
property watchdog_dio_inhibit_default: int
```

This register can be used to prevent the watchdog from changing specific IO. See DIO_INHIBIT for more information.

Register: WATCHDOG_DIO_INHIBIT_DEFAULT

Address: 61636

```
property watchdog_dio_state_default: int
```

The state high/low of the digital I/O after a Watchdog timeout. See DIO_STATE.

Register: WATCHDOG_DIO_STATE_DEFAULT

Address: 61632

```
property watchdog_enable_default: int
```

Write a 1 to enable the watchdog or a 0 to disable. The watchdog must be disabled before writing any of the other watchdog registers (except for WATCHDOG_STRICT_CLEAR).

Register: WATCHDOG_ENABLE_DEFAULT

Address: 61600

```
property watchdog_reset_enable_default: int
```

Timeout action: Set to 1 to enable device-reset on watchdog timeout.

Register: WATCHDOG_RESET_ENABLE_DEFAULT

Address: 61620

```
property watchdog_startup_delay_s_default: int
```

This specifies the initial timeout period at device bootup. This is used until the first time the watchdog is cleared or timeout ... after that the normal timeout is used.

Register: WATCHDOG_STARTUP_DELAY_S_DEFAULT

Address: 61606

```
property watchdog_strict_clear: int
```

When running in strict mode, writing the key to this register is the only way to clear the watchdog. Writing to this register while not using strict mode will clear the watchdog.

Register: WATCHDOG_STRICT_CLEAR

Address: 61614

```
property watchdog_strict_enable_default: int
```

Set to 1 to enable strict mode.

Register: WATCHDOG_STRICT_ENABLE_DEFAULT

Address: 61610

```
property watchdog_strict_key_default: int
```

When set to strict mode, this is the value that must be written to the clear register.

Register: WATCHDOG_STRICT_KEY_DEFAULT

Address: 61612

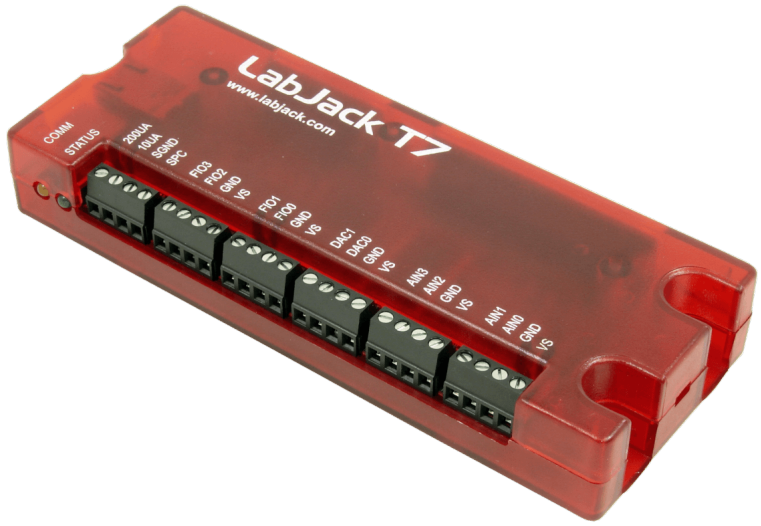
```
property watchdog_timeout_s_default: int
```

When the device receives any communication over USB/Ethernet/WiFi, the watchdog timer is cleared. If the watchdog timer is not cleared within the timeout period, the enabled actions will be done.

Register: WATCHDOG_TIMEOUT_S_DEFAULT

Address: 61604

LabJack T7/T7-Pro






```
class htf.labjack.T7(*, name: str | None = None, host: str | None = None, port: int = 502,
debuglevel: int = 0, timeout: float = 1.0, retries: int = 0)
```

LabJack adapter for the T7 and T7-Pro.

Parameters:

- **name** – If name is set the USB device's name is checked against it. If host is None a USB device is opened.
- **host** – If host is set a Modbus-TCP connection is opened. Name must not be set at the same time.
- **port** – The Modbus-TCP port to be used
- **debuglevel** – If set the Modbus packets are printed
- **timeout** – The timeout for every request
- **retries** – The number of retries for Modbus communication

```
property ain0: float
```

Returns the voltage of the specified analog input.

Register: AIN0 (AIN#(0:13)@0)

Address: 0

```
property ain0_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN0_BINARY (AIN#(0:13)_BINARY@0)

Address: 50000

```
property ain0_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@0)

Address: 9300

```
property ain0_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@0)

Address: 9600

```
property ain0_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@0)

Address: 9900

```
property ain0_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@0)

Address: 10200

```
property ain0_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@0)

Address: 10500

```
property ain0_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@0)

Address: 10800

```
property ain0_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@0)

Address: 11100

```
property ain0_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@0)

Address: 11400

```
property ain0_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@0)

Address: 11700

```
property ain0_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@0)

Address: 12000

```
property ain0_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN0_EF_INDEX (AIN#(0:149)_EF_INDEX@0)

Address: 9000

```
property ain0_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN0_EF_READ_A (AIN#(0:149)_EF_READ_A@0)

Address: 7000

```
property ain0_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN0_EF_READ_B (AIN#(0:149)_EF_READ_B@0)

Address: 7300

```
property ain0_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN0_EF_READ_C (AIN#(0:149)_EF_READ_C@0)

Address: 7600

```
property ain0_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN0_EF_READ_D (AIN#(0:149)_EF_READ_D@0)

Address: 7900

```
property ain0_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN0_NEGATIVE_CH (AIN#(0:13)_NEGATIVE_CH@0)

Address: 41000

```
property ain0_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN0_RANGE (AIN#(0:13)_RANGE@0)

Address: 40000

```
property ain0_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN0_RESOLUTION_INDEX (AIN#(0:13)_RESOLUTION_INDEX@0)

Address: 41500

```
property ain0_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN0_SETTLING_US (AIN#(0:13)_SETTLING_US@0)

Address: 42000

```
property ain1: float
```

Returns the voltage of the specified analog input.

Register: AIN1 (AIN#(0:13)@1)

Address: 2

```
property ain10: float
```

Returns the voltage of the specified analog input.

Register: AIN10 (AIN#(0:13)@10)

Address: 20

```
property ain100: float
```

Returns the voltage of the specified analog input.

Register: AIN100 (AIN#(48:127)@100)

Address: 200

```
property ain100_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN100_BINARY (AIN#(48:127)_BINARY@100)

Address: 50200

```
property ain100_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@100)

Address: 9500

```
property ain100_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@100)

Address: 9800

```
property ain100_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@100)

Address: 10100

```
property ain100_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@100)

Address: 10400

```
property ain100_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@100)

Address: 10700

```
property ain100_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@100)

Address: 11000

```
property ain100_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@100)

Address: 11300

```
property ain100_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@100)

Address: 11600

```
property ain100_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@100)

Address: 11900

```
property ain100_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@100)

Address: 12200

```
property ain100_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN100_EF_INDEX (AIN#(0:149)_EF_INDEX@100)

Address: 9200

```
property ain100_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN100_EF_READ_A (AIN#(0:149)_EF_READ_A@100)

Address: 7200

```
property ain100_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN100_EF_READ_B (AIN#(0:149)_EF_READ_B@100)

Address: 7500

```
property ain100_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN100_EF_READ_C (AIN#(0:149)_EF_READ_C@100)

Address: 7800

```
property ain100_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN100_EF_READ_D (AIN#(0:149)_EF_READ_D@100)

Address: 8100

```
property ain100_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN100_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@100)

Address: 41100

```
property ain100_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN100_RANGE (AIN#(48:127)_RANGE@100)

Address: 40200

```
property ain100_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN100_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@100)

Address: 41600

```
property ain100_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN100_SETTLING_US (AIN#(48:127)_SETTLING_US@100)

Address: 42200

```
property ain101: float
```

Returns the voltage of the specified analog input.

Register: AIN101 (AIN#(48:127)@101)

Address: 202

```
property ain101_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN101_BINARY (AIN#(48:127)_BINARY@101)

Address: 50202

```
property ain101_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@101)

Address: 9502

```
property ain101_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@101)

Address: 9802

```
property ain101_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@101)

Address: 10102

```
property ain101_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@101)

Address: 10402

```
property ain101_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@101)

Address: 10702

```
property ain101_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@101)

Address: 11002

```
property ain101_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@101)

Address: 11302

```
property ain101_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@101)

Address: 11602

```
property ain101_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@101)

Address: 11902

```
property ain101_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@101)

Address: 12202

```
property ain101_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN101_EF_INDEX (AIN#(0:149)_EF_INDEX@101)

Address: 9202

```
property ain101_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN101_EF_READ_A (AIN#(0:149)_EF_READ_A@101)

Address: 7202

```
property ain101_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN101_EF_READ_B (AIN#(0:149)_EF_READ_B@101)

Address: 7502

```
property ain101_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN101_EF_READ_C (AIN#(0:149)_EF_READ_C@101)

Address: 7802

```
property ain101_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN101_EF_READ_D (AIN#(0:149)_EF_READ_D@101)

Address: 8102

```
property ain101_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN101_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@101)

Address: 41101

```
property ain101_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN101_RANGE (AIN#(48:127)_RANGE@101)

Address: 40202

```
property ain101_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN101_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@101)

Address: 41601

```
property ain101_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN101_SETTLING_US (AIN#(48:127)_SETTLING_US@101)

Address: 42202

```
property ain102: float
```

Returns the voltage of the specified analog input.

Register: AIN102 (AIN#(48:127)@102)

Address: 204

```
property ain102_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN102_BINARY (AIN#(48:127)_BINARY@102)

Address: 50204

```
property ain102_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@102)

Address: 9504

```
property ain102_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@102)

Address: 9804

```
property ain102_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@102)

Address: 10104

```
property ain102_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@102)

Address: 10404

```
property ain102_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@102)

Address: 10704

```
property ain102_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@102)

Address: 11004

```
property ain102_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@102)

Address: 11304

```
property ain102_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@102)

Address: 11604

```
property ain102_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@102)

Address: 11904

```
property ain102_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@102)

Address: 12204

```
property ain102_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN102_EF_INDEX (AIN#(0:149)_EF_INDEX@102)

Address: 9204

```
property ain102_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN102_EF_READ_A (AIN#(0:149)_EF_READ_A@102)

Address: 7204

```
property ain102_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN102_EF_READ_B (AIN#(0:149)_EF_READ_B@102)

Address: 7504

```
property ain102_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN102_EF_READ_C (AIN#(0:149)_EF_READ_C@102)

Address: 7804

```
property ain102_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN102_EF_READ_D (AIN#(0:149)_EF_READ_D@102)

Address: 8104

```
property ain102_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN102_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@102)

Address: 41102

```
property ain102_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN102_RANGE (AIN#(48:127)_RANGE@102)

Address: 40204

```
property ain102_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN102_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@102)

Address: 41602

```
property ain102_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN102_SETTLING_US (AIN#(48:127)_SETTLING_US@102)

Address: 42204

```
property ain103: float
```

Returns the voltage of the specified analog input.

Register: AIN103 (AIN#(48:127)@103)

Address: 206

```
property ain103_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN103_BINARY (AIN#(48:127)_BINARY@103)

Address: 50206

```
property ain103_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@103)

Address: 9506

```
property ain103_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@103)

Address: 9806

```
property ain103_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@103)

Address: 10106

```
property ain103_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@103)

Address: 10406

```
property ain103_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@103)

Address: 10706

```
property ain103_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@103)

Address: 11006

```
property ain103_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@103)

Address: 11306

```
property ain103_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@103)

Address: 11606

```
property ain103_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@103)

Address: 11906

```
property ain103_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@103)

Address: 12206

```
property ain103_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN103_EF_INDEX (AIN#(0:149)_EF_INDEX@103)

Address: 9206

```
property ain103_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN103_EF_READ_A (AIN#(0:149)_EF_READ_A@103)

Address: 7206

```
property ain103_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN103_EF_READ_B (AIN#(0:149)_EF_READ_B@103)

Address: 7506

```
property ain103_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN103_EF_READ_C (AIN#(0:149)_EF_READ_C@103)

Address: 7806

```
property ain103_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN103_EF_READ_D (AIN#(0:149)_EF_READ_D@103)

Address: 8106

```
property ain103_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN103_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@103)

Address: 41103

```
property ain103_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN103_RANGE (AIN#(48:127)_RANGE@103)

Address: 40206

```
property ain103_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN103_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@103)

Address: 41603

```
property ain103_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN103_SETTLING_US (AIN#(48:127)_SETTLING_US@103)

Address: 42206

```
property ain104: float
```

Returns the voltage of the specified analog input.

Register: AIN104 (AIN#(48:127)@104)

Address: 208

```
property ain104_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN104_BINARY (AIN#(48:127)_BINARY@104)

Address: 50208

```
property ain104_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@104)

Address: 9508

```
property ain104_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@104)

Address: 9808

```
property ain104_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@104)

Address: 10108

```
property ain104_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@104)

Address: 10408

```
property ain104_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@104)

Address: 10708

```
property ain104_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@104)

Address: 11008

```
property ain104_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@104)

Address: 11308

```
property ain104_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@104)

Address: 11608

```
property ain104_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@104)

Address: 11908

```
property ain104_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@104)

Address: 12208

```
property ain104_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN104_EF_INDEX (AIN#(0:149)_EF_INDEX@104)

Address: 9208

```
property ain104_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN104_EF_READ_A (AIN#(0:149)_EF_READ_A@104)

Address: 7208

```
property ain104_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN104_EF_READ_B (AIN#(0:149)_EF_READ_B@104)

Address: 7508

```
property ain104_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN104_EF_READ_C (AIN#(0:149)_EF_READ_C@104)

Address: 7808

```
property ain104_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN104_EF_READ_D (AIN#(0:149)_EF_READ_D@104)

Address: 8108

```
property ain104_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN104_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@104)

Address: 41104

```
property ain104_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN104_RANGE (AIN#(48:127)_RANGE@104)

Address: 40208

```
property ain104_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN104_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@104)

Address: 41604

```
property ain104_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN104_SETTLING_US (AIN#(48:127)_SETTLING_US@104)

Address: 42208

```
property ain105: float
```

Returns the voltage of the specified analog input.

Register: AIN105 (AIN#(48:127)@105)

Address: 210

```
property ain105_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN105_BINARY (AIN#(48:127)_BINARY@105)

Address: 50210

```
property ain105_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@105)

Address: 9510

```
property ain105_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@105)

Address: 9810

```
property ain105_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@105)

Address: 10110

```
property ain105_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@105)

Address: 10410

```
property ain105_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@105)

Address: 10710

```
property ain105_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@105)

Address: 11010

```
property ain105_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@105)

Address: 11310

```
property ain105_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@105)

Address: 11610

```
property ain105_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@105)

Address: 11910

```
property ain105_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@105)

Address: 12210

```
property ain105_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN105_EF_INDEX (AIN#(0:149)_EF_INDEX@105)

Address: 9210

```
property ain105_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN105_EF_READ_A (AIN#(0:149)_EF_READ_A@105)

Address: 7210

```
property ain105_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN105_EF_READ_B (AIN#(0:149)_EF_READ_B@105)

Address: 7510

```
property ain105_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN105_EF_READ_C (AIN#(0:149)_EF_READ_C@105)

Address: 7810

```
property ain105_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN105_EF_READ_D (AIN#(0:149)_EF_READ_D@105)

Address: 8110

```
property ain105_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN105_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@105)

Address: 41105

```
property ain105_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN105_RANGE (AIN#(48:127)_RANGE@105)

Address: 40210

```
property ain105_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN105_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@105)

Address: 41605

```
property ain105_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN105_SETTLING_US (AIN#(48:127)_SETTLING_US@105)

Address: 42210

```
property ain106: float
```

Returns the voltage of the specified analog input.

Register: AIN106 (AIN#(48:127)@106)

Address: 212

```
property ain106_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN106_BINARY (AIN#(48:127)_BINARY@106)

Address: 50212

```
property ain106_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@106)

Address: 9512

```
property ain106_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@106)

Address: 9812

```
property ain106_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@106)

Address: 10112

```
property ain106_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@106)

Address: 10412

```
property ain106_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@106)

Address: 10712

```
property ain106_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@106)

Address: 11012

```
property ain106_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@106)

Address: 11312

```
property ain106_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@106)

Address: 11612

```
property ain106_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@106)

Address: 11912

```
property ain106_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@106)

Address: 12212

```
property ain106_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN106_EF_INDEX (AIN#(0:149)_EF_INDEX@106)

Address: 9212

```
property ain106_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN106_EF_READ_A (AIN#(0:149)_EF_READ_A@106)

Address: 7212

```
property ain106_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN106_EF_READ_B (AIN#(0:149)_EF_READ_B@106)

Address: 7512

```
property ain106_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN106_EF_READ_C (AIN#(0:149)_EF_READ_C@106)

Address: 7812

```
property ain106_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN106_EF_READ_D (AIN#(0:149)_EF_READ_D@106)

Address: 8112

```
property ain106_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN106_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@106)

Address: 41106

```
property ain106_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN106_RANGE (AIN#(48:127)_RANGE@106)

Address: 40212

```
property ain106_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN106_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@106)

Address: 41606

```
property ain106_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN106_SETTLING_US (AIN#(48:127)_SETTLING_US@106)

Address: 42212

```
property ain107: float
```

Returns the voltage of the specified analog input.

Register: AIN107 (AIN#(48:127)@107)

Address: 214

```
property ain107_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN107_BINARY (AIN#(48:127)_BINARY@107)

Address: 50214

```
property ain107_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@107)

Address: 9514

```
property ain107_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@107)

Address: 9814

```
property ain107_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@107)

Address: 10114

```
property ain107_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@107)

Address: 10414

```
property ain107_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@107)

Address: 10714

```
property ain107_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@107)

Address: 11014

```
property ain107_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@107)

Address: 11314

```
property ain107_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@107)

Address: 11614

```
property ain107_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@107)

Address: 11914

```
property ain107_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@107)

Address: 12214

```
property ain107_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T;

25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN107_EF_INDEX (AIN#(0:149)_EF_INDEX@107)

Address: 9214

```
property ain107_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN107_EF_READ_A (AIN#(0:149)_EF_READ_A@107)

Address: 7214

```
property ain107_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN107_EF_READ_B (AIN#(0:149)_EF_READ_B@107)

Address: 7514

```
property ain107_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN107_EF_READ_C (AIN#(0:149)_EF_READ_C@107)

Address: 7814

```
property ain107_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN107_EF_READ_D (AIN#(0:149)_EF_READ_D@107)

Address: 8114

```
property ain107_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN107_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@107)

Address: 41107

```
property ain107_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN107_RANGE (AIN#(48:127)_RANGE@107)

Address: 40214

```
property ain107_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN107_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@107)

Address: 41607

```
property ain107_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN107_SETTLING_US (AIN#(48:127)_SETTLING_US@107)

Address: 42214

```
property ain108: float
```

Returns the voltage of the specified analog input.

Register: AIN108 (AIN#(48:127)@108)

Address: 216

```
property ain108_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN108_BINARY (AIN#(48:127)_BINARY@108)

Address: 50216

```
property ain108_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@108)

Address: 9516

```
property ain108_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@108)

Address: 9816

```
property ain108_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@108)

Address: 10116

```
property ain108_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@108)

Address: 10416

```
property ain108_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@108)

Address: 10716

```
property ain108_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@108)

Address: 11016

```
property ain108_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@108)

Address: 11316

```
property ain108_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@108)

Address: 11616

```
property ain108_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@108)

Address: 11916

```
property ain108_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@108)

Address: 12216

```
property ain108_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN108_EF_INDEX (AIN#(0:149)_EF_INDEX@108)

Address: 9216

```
property ain108_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN108_EF_READ_A (AIN#(0:149)_EF_READ_A@108)

Address: 7216

```
property ain108_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN108_EF_READ_B (AIN#(0:149)_EF_READ_B@108)

Address: 7516

```
property ain108_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN108_EF_READ_C (AIN#(0:149)_EF_READ_C@108)

Address: 7816

```
property ain108_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN108_EF_READ_D (AIN#(0:149)_EF_READ_D@108)

Address: 8116

```
property ain108_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN108_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@108)

Address: 41108

```
property ain108_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN108_RANGE (AIN#(48:127)_RANGE@108)

Address: 40216

```
property ain108_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN108_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@108)

Address: 41608

```
property ain108_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN108_SETTLING_US (AIN#(48:127)_SETTLING_US@108)

Address: 42216

```
property ain109: float
```

Returns the voltage of the specified analog input.

Register: AIN109 (AIN#(48:127)@109)

Address: 218

```
property ain109_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN109_BINARY (AIN#(48:127)_BINARY@109)

Address: 50218

```
property ain109_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@109)

Address: 9518

```
property ain109_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@109)

Address: 9818

```
property ain109_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@109)

Address: 10118

```
property ain109_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@109)

Address: 10418

```
property ain109_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@109)

Address: 10718

```
property ain109_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@109)

Address: 11018

```
property ain109_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@109)

Address: 11318

```
property ain109_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@109)

Address: 11618

```
property ain109_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@109)

Address: 11918

```
property ain109_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@109)

Address: 12218

```
property ain109_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN109_EF_INDEX (AIN#(0:149)_EF_INDEX@109)

Address: 9218

```
property ain109_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN109_EF_READ_A (AIN#(0:149)_EF_READ_A@109)

Address: 7218

```
property ain109_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN109_EF_READ_B (AIN#(0:149)_EF_READ_B@109)

Address: 7518

```
property ain109_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN109_EF_READ_C (AIN#(0:149)_EF_READ_C@109)

Address: 7818

```
property ain109_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN109_EF_READ_D (AIN#(0:149)_EF_READ_D@109)

Address: 8118

```
property ain109_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN109_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@109)

Address: 41109

```
property ain109_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN109_RANGE (AIN#(48:127)_RANGE@109)

Address: 40218

```
property ain109_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN109_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@109)

Address: 41609

```
property ain109_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN109_SETTLING_US (AIN#(48:127)_SETTLING_US@109)

Address: 42218

```
property ain10_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN10_BINARY (AIN#(0:13)_BINARY@10)

Address: 50020

```
property ain10_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@10)

Address: 9320

```
property ain10_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@10)

Address: 9620

```
property ain10_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@10)

Address: 9920

```
property ain10_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@10)

Address: 10220

```
property ain10_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@10)

Address: 10520

```
property ain10_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@10)

Address: 10820

```
property ain10_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@10)

Address: 11120

```
property ain10_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@10)

Address: 11420

```
property ain10_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@10)

Address: 11720

```
property ain10_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@10)

Address: 12020

```
property ain10_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN10_EF_INDEX (AIN#(0:149)_EF_INDEX@10)

Address: 9020

```
property ain10_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN10_EF_READ_A (AIN#(0:149)_EF_READ_A@10)

Address: 7020

```
property ain10_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN10_EF_READ_B (AIN#(0:149)_EF_READ_B@10)

Address: 7320

```
property ain10_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN10_EF_READ_C (AIN#(0:149)_EF_READ_C@10)

Address: 7620

```
property ain10_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN10_EF_READ_D (AIN#(0:149)_EF_READ_D@10)

Address: 7920

```
property ain10_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN10_NEGATIVE_CH (AIN#(0:13)_NEGATIVE_CH@10)

Address: 41010

```
property ain10_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN10_RANGE (AIN#(0:13)_RANGE@10)

Address: 40020

```
property ain10_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN10_RESOLUTION_INDEX (AIN#(0:13)_RESOLUTION_INDEX@10)

Address: 41510

```
property ain10_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN10_SETTLING_US (AIN#(0:13)_SETTLING_US@10)

Address: 42020

```
property ain11: float
```

Returns the voltage of the specified analog input.

Register: AIN11 (AIN#(0:13)@11)

Address: 22

```
property ain110: float
```

Returns the voltage of the specified analog input.

Register: AIN110 (AIN#(48:127)@110)

Address: 220

```
property ain110_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN110_BINARY (AIN#(48:127)_BINARY@110)

Address: 50220

```
property ain110_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@110)

Address: 9520

```
property ain110_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@110)

Address: 9820

```
property ain110_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@110)

Address: 10120

```
property ain110_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@110)

Address: 10420

```
property ain110_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@110)

Address: 10720

```
property ain110_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@110)

Address: 11020

```
property ain110_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@110)

Address: 11320

```
property ain110_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@110)

Address: 11620

```
property ain110_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@110)

Address: 11920

```
property ain110_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@110)

Address: 12220

```
property ain110_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN110_EF_INDEX (AIN#(0:149)_EF_INDEX@110)

Address: 9220

```
property ain110_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN110_EF_READ_A (AIN#(0:149)_EF_READ_A@110)

Address: 7220

```
property ain110_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN110_EF_READ_B (AIN#(0:149)_EF_READ_B@110)

Address: 7520

```
property ain110_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN110_EF_READ_C (AIN#(0:149)_EF_READ_C@110)

Address: 7820

```
property ain110_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN110_EF_READ_D (AIN#(0:149)_EF_READ_D@110)

Address: 8120

```
property ain110_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN110_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@110)

Address: 41110

```
property ain110_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN110_RANGE (AIN#(48:127)_RANGE@110)

Address: 40220

```
property ain110_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN110_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@110)

Address: 41610

```
property ain110_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN110_SETTLING_US (AIN#(48:127)_SETTLING_US@110)

Address: 42220

```
property ain111: float
```

Returns the voltage of the specified analog input.

Register: AIN111 (AIN#(48:127)@111)

Address: 222

```
property ain111_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN111_BINARY (AIN#(48:127)_BINARY@111)

Address: 50222

```
property ain111_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@111)

Address: 9522

```
property ain111_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@111)

Address: 9822

```
property ain111_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@111)

Address: 10122

```
property ain111_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@111)

Address: 10422

```
property ain111_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@111)

Address: 10722

```
property ain111_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@111)

Address: 11022

```
property ain111_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@111)

Address: 11322

```
property ain111_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@111)

Address: 11622

```
property ain111_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@111)

Address: 11922

```
property ain111_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@111)

Address: 12222

```
property ain111_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN111_EF_INDEX (AIN#(0:149)_EF_INDEX@111)

Address: 9222

```
property ain111_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN111_EF_READ_A (AIN#(0:149)_EF_READ_A@111)

Address: 7222

```
property ain111_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN111_EF_READ_B (AIN#(0:149)_EF_READ_B@111)

Address: 7522

```
property ain111_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN111_EF_READ_C (AIN#(0:149)_EF_READ_C@111)

Address: 7822

```
property ain111_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN111_EF_READ_D (AIN#(0:149)_EF_READ_D@111)

Address: 8122

```
property ain111_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN111_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@111)

Address: 41111

```
property ain111_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN111_RANGE (AIN#(48:127)_RANGE@111)

Address: 40222

```
property ain111_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN111_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@111)

Address: 41611

```
property ain111_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN111_SETTLING_US (AIN#(48:127)_SETTLING_US@111)

Address: 42222

```
property ain112: float
```

Returns the voltage of the specified analog input.

Register: AIN112 (AIN#(48:127)@112)

Address: 224

```
property ain112_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN112_BINARY (AIN#(48:127)_BINARY@112)

Address: 50224

```
property ain112_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@112)

Address: 9524

```
property ain112_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@112)

Address: 9824

```
property ain112_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@112)

Address: 10124

```
property ain112_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@112)

Address: 10424

```
property ain112_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@112)

Address: 10724

```
property ain112_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@112)

Address: 11024

```
property ain112_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@112)

Address: 11324

```
property ain112_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@112)

Address: 11624

```
property ain112_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@112)

Address: 11924

```
property ain112_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@112)

Address: 12224

```
property ain112_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN112_EF_INDEX (AIN#(0:149)_EF_INDEX@112)

Address: 9224

```
property ain112_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN112_EF_READ_A (AIN#(0:149)_EF_READ_A@112)

Address: 7224

```
property ain112_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN112_EF_READ_B (AIN#(0:149)_EF_READ_B@112)

Address: 7524

```
property ain112_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN112_EF_READ_C (AIN#(0:149)_EF_READ_C@112)

Address: 7824

```
property ain112_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN112_EF_READ_D (AIN#(0:149)_EF_READ_D@112)

Address: 8124

```
property ain112_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN112_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@112)

Address: 41112

```
property ain112_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN112_RANGE (AIN#(48:127)_RANGE@112)

Address: 40224

```
property ain112_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN112_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@112)

Address: 41612

```
property ain112_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN112_SETTLING_US (AIN#(48:127)_SETTLING_US@112)

Address: 42224

```
property ain113: float
```

Returns the voltage of the specified analog input.

Register: AIN113 (AIN#(48:127)@113)

Address: 226

```
property ain113_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN113_BINARY (AIN#(48:127)_BINARY@113)

Address: 50226

```
property ain113_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@113)

Address: 9526

```
property ain113_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@113)

Address: 9826

```
property ain113_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@113)

Address: 10126

```
property ain113_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@113)

Address: 10426

```
property ain113_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@113)

Address: 10726

```
property ain113_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@113)

Address: 11026

```
property ain113_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@113)

Address: 11326

```
property ain113_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@113)

Address: 11626

```
property ain113_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@113)

Address: 11926

```
property ain113_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@113)

Address: 12226

```
property ain113_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN113_EF_INDEX (AIN#(0:149)_EF_INDEX@113)

Address: 9226

```
property ain113_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN113_EF_READ_A (AIN#(0:149)_EF_READ_A@113)

Address: 7226

```
property ain113_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN113_EF_READ_B (AIN#(0:149)_EF_READ_B@113)

Address: 7526

```
property ain113_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN113_EF_READ_C (AIN#(0:149)_EF_READ_C@113)

Address: 7826

```
property ain113_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN113_EF_READ_D (AIN#(0:149)_EF_READ_D@113)

Address: 8126

```
property ain113_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN113_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@113)

Address: 41113

```
property ain113_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN113_RANGE (AIN#(48:127)_RANGE@113)

Address: 40226

```
property ain113_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN113_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@113)

Address: 41613

```
property ain113_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN113_SETTLING_US (AIN#(48:127)_SETTLING_US@113)

Address: 42226

```
property ain114: float
```

Returns the voltage of the specified analog input.

Register: AIN114 (AIN#(48:127)@114)

Address: 228

```
property ain114_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN114_BINARY (AIN#(48:127)_BINARY@114)

Address: 50228

```
property ain114_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@114)

Address: 9528

```
property ain114_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@114)

Address: 9828

```
property ain114_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@114)

Address: 10128

```
property ain114_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@114)

Address: 10428

```
property ain114_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@114)

Address: 10728

```
property ain114_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@114)

Address: 11028

```
property ain114_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@114)

Address: 11328

```
property ain114_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@114)

Address: 11628

```
property ain114_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@114)

Address: 11928

```
property ain114_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@114)

Address: 12228

```
property ain114_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN114_EF_INDEX (AIN#(0:149)_EF_INDEX@114)

Address: 9228

```
property ain114_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN114_EF_READ_A (AIN#(0:149)_EF_READ_A@114)

Address: 7228

```
property ain114_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN114_EF_READ_B (AIN#(0:149)_EF_READ_B@114)

Address: 7528

```
property ain114_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN114_EF_READ_C (AIN#(0:149)_EF_READ_C@114)

Address: 7828

```
property ain114_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN114_EF_READ_D (AIN#(0:149)_EF_READ_D@114)

Address: 8128

```
property ain114_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN114_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@114)

Address: 41114

```
property ain114_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN114_RANGE (AIN#(48:127)_RANGE@114)

Address: 40228

```
property ain114_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN114_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@114)

Address: 41614

```
property ain114_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN114_SETTLING_US (AIN#(48:127)_SETTLING_US@114)

Address: 42228

```
property ain115: float
```

Returns the voltage of the specified analog input.

Register: AIN115 (AIN#(48:127)@115)

Address: 230

```
property ain115_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN115_BINARY (AIN#(48:127)_BINARY@115)

Address: 50230

```
property ain115_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@115)

Address: 9530

```
property ain115_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@115)

Address: 9830

```
property ain115_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@115)

Address: 10130

```
property ain115_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@115)

Address: 10430

```
property ain115_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@115)

Address: 10730

```
property ain115_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@115)

Address: 11030

```
property ain115_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@115)

Address: 11330

```
property ain115_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@115)

Address: 11630

```
property ain115_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@115)

Address: 11930

```
property ain115_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@115)

Address: 12230

```
property ain115_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN115_EF_INDEX (AIN#(0:149)_EF_INDEX@115)

Address: 9230

```
property ain115_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN115_EF_READ_A (AIN#(0:149)_EF_READ_A@115)

Address: 7230

```
property ain115_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN115_EF_READ_B (AIN#(0:149)_EF_READ_B@115)

Address: 7530

```
property ain115_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN115_EF_READ_C (AIN#(0:149)_EF_READ_C@115)

Address: 7830

```
property ain115_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN115_EF_READ_D (AIN#(0:149)_EF_READ_D@115)

Address: 8130

```
property ain115_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN115_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@115)

Address: 41115

```
property ain115_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN115_RANGE (AIN#(48:127)_RANGE@115)

Address: 40230

```
property ain115_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN115_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@115)

Address: 41615

```
property ain115_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN115_SETTLING_US (AIN#(48:127)_SETTLING_US@115)

Address: 42230

```
property ain116: float
```

Returns the voltage of the specified analog input.

Register: AIN116 (AIN#(48:127)@116)

Address: 232

```
property ain116_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN116_BINARY (AIN#(48:127)_BINARY@116)

Address: 50232

```
property ain116_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@116)

Address: 9532

```
property ain116_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@116)

Address: 9832

```
property ain116_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@116)

Address: 10132

```
property ain116_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@116)

Address: 10432

```
property ain116_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@116)

Address: 10732

```
property ain116_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@116)

Address: 11032

```
property ain116_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@116)

Address: 11332

```
property ain116_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@116)

Address: 11632

```
property ain116_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@116)

Address: 11932

```
property ain116_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@116)

Address: 12232

```
property ain116_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN116_EF_INDEX (AIN#(0:149)_EF_INDEX@116)

Address: 9232

```
property ain116_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN116_EF_READ_A (AIN#(0:149)_EF_READ_A@116)

Address: 7232

```
property ain116_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN116_EF_READ_B (AIN#(0:149)_EF_READ_B@116)

Address: 7532

```
property ain116_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN116_EF_READ_C (AIN#(0:149)_EF_READ_C@116)

Address: 7832

```
property ain116_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN116_EF_READ_D (AIN#(0:149)_EF_READ_D@116)

Address: 8132

```
property ain116_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN116_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@116)

Address: 41116

```
property ain116_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN116_RANGE (AIN#(48:127)_RANGE@116)

Address: 40232

```
property ain116_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN116_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@116)

Address: 41616

```
property ain116_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN116_SETTLING_US (AIN#(48:127)_SETTLING_US@116)

Address: 42232

```
property ain117: float
```

Returns the voltage of the specified analog input.

Register: AIN117 (AIN#(48:127)@117)

Address: 234

```
property ain117_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN117_BINARY (AIN#(48:127)_BINARY@117)

Address: 50234

```
property ain117_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@117)

Address: 9534

```
property ain117_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@117)

Address: 9834

```
property ain117_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@117)

Address: 10134

```
property ain117_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@117)

Address: 10434

```
property ain117_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@117)

Address: 10734

```
property ain117_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@117)

Address: 11034

```
property ain117_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@117)

Address: 11334

```
property ain117_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@117)

Address: 11634

```
property ain117_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@117)

Address: 11934

```
property ain117_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@117)

Address: 12234

```
property ain117_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN117_EF_INDEX (AIN#(0:149)_EF_INDEX@117)

Address: 9234

```
property ain117_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN117_EF_READ_A (AIN#(0:149)_EF_READ_A@117)

Address: 7234

```
property ain117_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN117_EF_READ_B (AIN#(0:149)_EF_READ_B@117)

Address: 7534

```
property ain117_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN117_EF_READ_C (AIN#(0:149)_EF_READ_C@117)

Address: 7834

```
property ain117_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN117_EF_READ_D (AIN#(0:149)_EF_READ_D@117)

Address: 8134

```
property ain117_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN117_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@117)

Address: 41117

```
property ain117_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN117_RANGE (AIN#(48:127)_RANGE@117)

Address: 40234

```
property ain117_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN117_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@117)

Address: 41617

```
property ain117_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN117_SETTLING_US (AIN#(48:127)_SETTLING_US@117)

Address: 42234

```
property ain118: float
```

Returns the voltage of the specified analog input.

Register: AIN118 (AIN#(48:127)@118)

Address: 236

```
property ain118_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN118_BINARY (AIN#(48:127)_BINARY@118)

Address: 50236

```
property ain118_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@118)

Address: 9536

```
property ain118_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@118)

Address: 9836

```
property ain118_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@118)

Address: 10136

```
property ain118_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@118)

Address: 10436

```
property ain118_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@118)

Address: 10736

```
property ain118_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@118)

Address: 11036

```
property ain118_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@118)

Address: 11336

```
property ain118_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@118)

Address: 11636

```
property ain118_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@118)

Address: 11936

```
property ain118_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@118)

Address: 12236

```
property ain118_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T;

25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN118_EF_INDEX (AIN#(0:149)_EF_INDEX@118)

Address: 9236

```
property ain118_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN118_EF_READ_A (AIN#(0:149)_EF_READ_A@118)

Address: 7236

```
property ain118_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN118_EF_READ_B (AIN#(0:149)_EF_READ_B@118)

Address: 7536

```
property ain118_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN118_EF_READ_C (AIN#(0:149)_EF_READ_C@118)

Address: 7836

```
property ain118_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN118_EF_READ_D (AIN#(0:149)_EF_READ_D@118)

Address: 8136

```
property ain118_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN118_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@118)

Address: 41118

```
property ain118_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN118_RANGE (AIN#(48:127)_RANGE@118)

Address: 40236

```
property ain118_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN118_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@118)

Address: 41618

```
property ain118_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN118_SETTLING_US (AIN#(48:127)_SETTLING_US@118)

Address: 42236

```
property ain119: float
```

Returns the voltage of the specified analog input.

Register: AIN119 (AIN#(48:127)@119)

Address: 238

```
property ain119_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN119_BINARY (AIN#(48:127)_BINARY@119)

Address: 50238

```
property ain119_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@119)

Address: 9538

```
property ain119_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@119)

Address: 9838

```
property ain119_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@119)

Address: 10138

```
property ain119_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@119)

Address: 10438

```
property ain119_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@119)

Address: 10738

```
property ain119_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@119)

Address: 11038

```
property ain119_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@119)

Address: 11338

```
property ain119_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@119)

Address: 11638

```
property ain119_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@119)

Address: 11938

```
property ain119_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@119)

Address: 12238

```
property ain119_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN119_EF_INDEX (AIN#(0:149)_EF_INDEX@119)

Address: 9238

```
property ain119_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN119_EF_READ_A (AIN#(0:149)_EF_READ_A@119)

Address: 7238

```
property ain119_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN119_EF_READ_B (AIN#(0:149)_EF_READ_B@119)

Address: 7538

```
property ain119_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN119_EF_READ_C (AIN#(0:149)_EF_READ_C@119)

Address: 7838

```
property ain119_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN119_EF_READ_D (AIN#(0:149)_EF_READ_D@119)

Address: 8138

```
property ain119_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN119_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@119)

Address: 41119

```
property ain119_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN119_RANGE (AIN#(48:127)_RANGE@119)

Address: 40238

```
property ain119_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN119_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@119)

Address: 41619

```
property ain119_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN119_SETTLING_US (AIN#(48:127)_SETTLING_US@119)

Address: 42238

```
property ain11_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN11_BINARY (AIN#(0:13)_BINARY@11)

Address: 50022

```
property ain11_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@11)

Address: 9322

```
property ain11_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@11)

Address: 9622

```
property ain11_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@11)

Address: 9922

```
property ain11_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@11)

Address: 10222

```
property ain11_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@11)

Address: 10522

```
property ain11_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@11)

Address: 10822

```
property ain11_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@11)

Address: 11122

```
property ain11_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@11)

Address: 11422

```
property ain11_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@11)

Address: 11722

```
property ain11_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@11)

Address: 12022

```
property ain11_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN11_EF_INDEX (AIN#(0:149)_EF_INDEX@11)

Address: 9022

```
property ain11_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN11_EF_READ_A (AIN#(0:149)_EF_READ_A@11)

Address: 7022

```
property ain11_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN11_EF_READ_B (AIN#(0:149)_EF_READ_B@11)

Address: 7322

```
property ain11_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN11_EF_READ_C (AIN#(0:149)_EF_READ_C@11)

Address: 7622

```
property ain11_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN11_EF_READ_D (AIN#(0:149)_EF_READ_D@11)

Address: 7922

```
property ain11_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN11_NEGATIVE_CH (AIN#(0:13)_NEGATIVE_CH@11)

Address: 41011

```
property ain11_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN11_RANGE (AIN#(0:13)_RANGE@11)

Address: 40022

```
property ain11_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN11_RESOLUTION_INDEX (AIN#(0:13)_RESOLUTION_INDEX@11)

Address: 41511

```
property ain11_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN11_SETTLING_US (AIN#(0:13)_SETTLING_US@11)

Address: 42022

```
property ain12: float
```

Returns the voltage of the specified analog input.

Register: AIN12 (AIN#(0:13)@12)

Address: 24

```
property ain120: float
```

Returns the voltage of the specified analog input.

Register: AIN120 (AIN#(48:127)@120)

Address: 240

```
property ain120_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN120_BINARY (AIN#(48:127)_BINARY@120)

Address: 50240

```
property ain120_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@120)

Address: 9540

```
property ain120_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@120)

Address: 9840

```
property ain120_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@120)

Address: 10140

```
property ain120_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@120)

Address: 10440

```
property ain120_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@120)

Address: 10740

```
property ain120_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@120)

Address: 11040

```
property ain120_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@120)

Address: 11340

```
property ain120_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@120)

Address: 11640

```
property ain120_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@120)

Address: 11940

```
property ain120_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@120)

Address: 12240

```
property ain120_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN120_EF_INDEX (AIN#(0:149)_EF_INDEX@120)

Address: 9240

```
property ain120_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN120_EF_READ_A (AIN#(0:149)_EF_READ_A@120)

Address: 7240

```
property ain120_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN120_EF_READ_B (AIN#(0:149)_EF_READ_B@120)

Address: 7540

```
property ain120_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN120_EF_READ_C (AIN#(0:149)_EF_READ_C@120)

Address: 7840

```
property ain120_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN120_EF_READ_D (AIN#(0:149)_EF_READ_D@120)

Address: 8140

```
property ain120_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN120_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@120)

Address: 41120

```
property ain120_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN120_RANGE (AIN#(48:127)_RANGE@120)

Address: 40240

```
property ain120_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN120_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@120)

Address: 41620

```
property ain120_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN120_SETTLING_US (AIN#(48:127)_SETTLING_US@120)

Address: 42240

```
property ain121: float
```

Returns the voltage of the specified analog input.

Register: AIN121 (AIN#(48:127)@121)

Address: 242

```
property ain121_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN121_BINARY (AIN#(48:127)_BINARY@121)

Address: 50242

```
property ain121_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@121)

Address: 9542

```
property ain121_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@121)

Address: 9842

```
property ain121_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@121)

Address: 10142

```
property ain121_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@121)

Address: 10442

```
property ain121_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@121)

Address: 10742

```
property ain121_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@121)

Address: 11042

```
property ain121_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@121)

Address: 11342

```
property ain121_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@121)

Address: 11642

```
property ain121_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@121)

Address: 11942

```
property ain121_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@121)

Address: 12242

```
property ain121_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN121_EF_INDEX (AIN#(0:149)_EF_INDEX@121)

Address: 9242

```
property ain121_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN121_EF_READ_A (AIN#(0:149)_EF_READ_A@121)

Address: 7242

```
property ain121_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN121_EF_READ_B (AIN#(0:149)_EF_READ_B@121)

Address: 7542

```
property ain121_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN121_EF_READ_C (AIN#(0:149)_EF_READ_C@121)

Address: 7842

```
property ain121_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN121_EF_READ_D (AIN#(0:149)_EF_READ_D@121)

Address: 8142

```
property ain121_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN121_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@121)

Address: 41121

```
property ain121_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN121_RANGE (AIN#(48:127)_RANGE@121)

Address: 40242

```
property ain121_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN121_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@121)

Address: 41621

```
property ain121_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN121_SETTLING_US (AIN#(48:127)_SETTLING_US@121)

Address: 42242

```
property ain122: float
```

Returns the voltage of the specified analog input.

Register: AIN122 (AIN#(48:127)@122)

Address: 244

```
property ain122_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN122_BINARY (AIN#(48:127)_BINARY@122)

Address: 50244

```
property ain122_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@122)

Address: 9544

```
property ain122_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@122)

Address: 9844

```
property ain122_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@122)

Address: 10144

```
property ain122_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@122)

Address: 10444

```
property ain122_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@122)

Address: 10744

```
property ain122_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@122)

Address: 11044

```
property ain122_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@122)

Address: 11344

```
property ain122_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@122)

Address: 11644

```
property ain122_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@122)

Address: 11944

```
property ain122_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@122)

Address: 12244

```
property ain122_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN122_EF_INDEX (AIN#(0:149)_EF_INDEX@122)

Address: 9244

```
property ain122_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN122_EF_READ_A (AIN#(0:149)_EF_READ_A@122)

Address: 7244

```
property ain122_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN122_EF_READ_B (AIN#(0:149)_EF_READ_B@122)

Address: 7544

```
property ain122_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN122_EF_READ_C (AIN#(0:149)_EF_READ_C@122)

Address: 7844

```
property ain122_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN122_EF_READ_D (AIN#(0:149)_EF_READ_D@122)

Address: 8144

```
property ain122_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN122_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@122)

Address: 41122

```
property ain122_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN122_RANGE (AIN#(48:127)_RANGE@122)

Address: 40244

```
property ain122_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN122_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@122)

Address: 41622

```
property ain122_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN122_SETTLING_US (AIN#(48:127)_SETTLING_US@122)

Address: 42244

```
property ain123: float
```

Returns the voltage of the specified analog input.

Register: AIN123 (AIN#(48:127)@123)

Address: 246

```
property ain123_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN123_BINARY (AIN#(48:127)_BINARY@123)

Address: 50246

```
property ain123_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@123)

Address: 9546

```
property ain123_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@123)

Address: 9846

```
property ain123_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@123)

Address: 10146

```
property ain123_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@123)

Address: 10446

```
property ain123_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@123)

Address: 10746

```
property ain123_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@123)

Address: 11046

```
property ain123_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@123)

Address: 11346

```
property ain123_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@123)

Address: 11646

```
property ain123_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@123)

Address: 11946

```
property ain123_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@123)

Address: 12246

```
property ain123_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN123_EF_INDEX (AIN#(0:149)_EF_INDEX@123)

Address: 9246

```
property ain123_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN123_EF_READ_A (AIN#(0:149)_EF_READ_A@123)

Address: 7246

```
property ain123_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN123_EF_READ_B (AIN#(0:149)_EF_READ_B@123)

Address: 7546

```
property ain123_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN123_EF_READ_C (AIN#(0:149)_EF_READ_C@123)

Address: 7846

```
property ain123_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN123_EF_READ_D (AIN#(0:149)_EF_READ_D@123)

Address: 8146

```
property ain123_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN123_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@123)

Address: 41123

```
property ain123_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN123_RANGE (AIN#(48:127)_RANGE@123)

Address: 40246

```
property ain123_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN123_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@123)

Address: 41623

```
property ain123_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN123_SETTLING_US (AIN#(48:127)_SETTLING_US@123)

Address: 42246

```
property ain124: float
```

Returns the voltage of the specified analog input.

Register: AIN124 (AIN#(48:127)@124)

Address: 248

```
property ain124_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN124_BINARY (AIN#(48:127)_BINARY@124)

Address: 50248

```
property ain124_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@124)

Address: 9548

```
property ain124_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@124)

Address: 9848

```
property ain124_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@124)

Address: 10148

```
property ain124_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@124)

Address: 10448

```
property ain124_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@124)

Address: 10748

```
property ain124_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@124)

Address: 11048

```
property ain124_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@124)

Address: 11348

```
property ain124_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@124)

Address: 11648

```
property ain124_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@124)

Address: 11948

```
property ain124_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@124)

Address: 12248

```
property ain124_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN124_EF_INDEX (AIN#(0:149)_EF_INDEX@124)

Address: 9248

```
property ain124_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN124_EF_READ_A (AIN#(0:149)_EF_READ_A@124)

Address: 7248

```
property ain124_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN124_EF_READ_B (AIN#(0:149)_EF_READ_B@124)

Address: 7548

```
property ain124_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN124_EF_READ_C (AIN#(0:149)_EF_READ_C@124)

Address: 7848

```
property ain124_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN124_EF_READ_D (AIN#(0:149)_EF_READ_D@124)

Address: 8148

```
property ain124_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN124_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@124)

Address: 41124

```
property ain124_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN124_RANGE (AIN#(48:127)_RANGE@124)

Address: 40248

```
property ain124_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN124_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@124)

Address: 41624

```
property ain124_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN124_SETTLING_US (AIN#(48:127)_SETTLING_US@124)

Address: 42248

```
property ain125: float
```

Returns the voltage of the specified analog input.

Register: AIN125 (AIN#(48:127)@125)

Address: 250

```
property ain125_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN125_BINARY (AIN#(48:127)_BINARY@125)

Address: 50250

```
property ain125_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@125)

Address: 9550

```
property ain125_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@125)

Address: 9850

```
property ain125_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@125)

Address: 10150

```
property ain125_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@125)

Address: 10450

```
property ain125_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@125)

Address: 10750

```
property ain125_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@125)

Address: 11050

```
property ain125_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@125)

Address: 11350

```
property ain125_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@125)

Address: 11650

```
property ain125_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@125)

Address: 11950

```
property ain125_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@125)

Address: 12250

```
property ain125_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN125_EF_INDEX (AIN#(0:149)_EF_INDEX@125)

Address: 9250

```
property ain125_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN125_EF_READ_A (AIN#(0:149)_EF_READ_A@125)

Address: 7250

```
property ain125_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN125_EF_READ_B (AIN#(0:149)_EF_READ_B@125)

Address: 7550

```
property ain125_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN125_EF_READ_C (AIN#(0:149)_EF_READ_C@125)

Address: 7850

```
property ain125_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN125_EF_READ_D (AIN#(0:149)_EF_READ_D@125)

Address: 8150

```
property ain125_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN125_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@125)

Address: 41125

```
property ain125_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN125_RANGE (AIN#(48:127)_RANGE@125)

Address: 40250

```
property ain125_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN125_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@125)

Address: 41625

```
property ain125_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN125_SETTLING_US (AIN#(48:127)_SETTLING_US@125)

Address: 42250

```
property ain126: float
```

Returns the voltage of the specified analog input.

Register: AIN126 (AIN#(48:127)@126)

Address: 252

```
property ain126_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN126_BINARY (AIN#(48:127)_BINARY@126)

Address: 50252

```
property ain126_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@126)

Address: 9552

```
property ain126_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@126)

Address: 9852

```
property ain126_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@126)

Address: 10152

```
property ain126_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@126)

Address: 10452

```
property ain126_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@126)

Address: 10752

```
property ain126_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@126)

Address: 11052

```
property ain126_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@126)

Address: 11352

```
property ain126_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@126)

Address: 11652

```
property ain126_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@126)

Address: 11952

```
property ain126_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@126)

Address: 12252

```
property ain126_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN126_EF_INDEX (AIN#(0:149)_EF_INDEX@126)

Address: 9252

```
property ain126_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN126_EF_READ_A (AIN#(0:149)_EF_READ_A@126)

Address: 7252

```
property ain126_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN126_EF_READ_B (AIN#(0:149)_EF_READ_B@126)

Address: 7552

```
property ain126_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN126_EF_READ_C (AIN#(0:149)_EF_READ_C@126)

Address: 7852

```
property ain126_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN126_EF_READ_D (AIN#(0:149)_EF_READ_D@126)

Address: 8152

```
property ain126_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN126_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@126)

Address: 41126

```
property ain126_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN126_RANGE (AIN#(48:127)_RANGE@126)

Address: 40252

```
property ain126_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN126_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@126)

Address: 41626

```
property ain126_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN126_SETTLING_US (AIN#(48:127)_SETTLING_US@126)

Address: 42252

```
property ain127: float
```

Returns the voltage of the specified analog input.

Register: AIN127 (AIN#(48:127)@127)

Address: 254

```
property ain127_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN127_BINARY (AIN#(48:127)_BINARY@127)

Address: 50254

```
property ain127_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@127)

Address: 9554

```
property ain127_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@127)

Address: 9854

```
property ain127_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@127)

Address: 10154

```
property ain127_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@127)

Address: 10454

```
property ain127_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@127)

Address: 10754

```
property ain127_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@127)

Address: 11054

```
property ain127_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@127)

Address: 11354

```
property ain127_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@127)

Address: 11654

```
property ain127_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@127)

Address: 11954

```
property ain127_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@127)

Address: 12254

```
property ain127_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN127_EF_INDEX (AIN#(0:149)_EF_INDEX@127)

Address: 9254

```
property ain127_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN127_EF_READ_A (AIN#(0:149)_EF_READ_A@127)

Address: 7254

```
property ain127_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN127_EF_READ_B (AIN#(0:149)_EF_READ_B@127)

Address: 7554

```
property ain127_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN127_EF_READ_C (AIN#(0:149)_EF_READ_C@127)

Address: 7854

```
property ain127_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN127_EF_READ_D (AIN#(0:149)_EF_READ_D@127)

Address: 8154

```
property ain127_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN127_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@127)

Address: 41127

```
property ain127_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN127_RANGE (AIN#(48:127)_RANGE@127)

Address: 40254

```
property ain127_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN127_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@127)

Address: 41627

```
property ain127_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN127_SETTLING_US (AIN#(48:127)_SETTLING_US@127)

Address: 42254

```
property ain128_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@128)

Address: 9556

```
property ain128_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@128)

Address: 9856

```
property ain128_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@128)

Address: 10156

```
property ain128_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@128)

Address: 10456

```
property ain128_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@128)

Address: 10756

```
property ain128_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@128)

Address: 11056

```
property ain128_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@128)

Address: 11356

```
property ain128_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@128)

Address: 11656

```
property ain128_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@128)

Address: 11956

```
property ain128_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@128)

Address: 12256

```
property ain128_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN128_EF_INDEX (AIN#(0:149)_EF_INDEX@128)

Address: 9256

```
property ain128_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN128_EF_READ_A (AIN#(0:149)_EF_READ_A@128)

Address: 7256

```
property ain128_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN128_EF_READ_B (AIN#(0:149)_EF_READ_B@128)

Address: 7556

```
property ain128_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN128_EF_READ_C (AIN#(0:149)_EF_READ_C@128)

Address: 7856

```
property ain128_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN128_EF_READ_D (AIN#(0:149)_EF_READ_D@128)

Address: 8156

```
property ain129_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@129)

Address: 9558

```
property ain129_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@129)

Address: 9858

```
property ain129_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@129)

Address: 10158

```
property ain129_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@129)

Address: 10458

```
property ain129_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@129)

Address: 10758

```
property ain129_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@129)

Address: 11058

```
property ain129_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@129)

Address: 11358

```
property ain129_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@129)

Address: 11658

```
property ain129_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@129)

Address: 11958

```
property ain129_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@129)

Address: 12258

```
property ain129_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN129_EF_INDEX (AIN#(0:149)_EF_INDEX@129)

Address: 9258

```
property ain129_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN129_EF_READ_A (AIN#(0:149)_EF_READ_A@129)

Address: 7258

```
property ain129_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN129_EF_READ_B (AIN#(0:149)_EF_READ_B@129)

Address: 7558

```
property ain129_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN129_EF_READ_C (AIN#(0:149)_EF_READ_C@129)

Address: 7858

```
property ain129_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN129_EF_READ_D (AIN#(0:149)_EF_READ_D@129)

Address: 8158

```
property ain12_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN12_BINARY (AIN#(0:13)_BINARY@12)

Address: 50024

```
property ain12_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@12)

Address: 9324

```
property ain12_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@12)

Address: 9624

```
property ain12_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@12)

Address: 9924

```
property ain12_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@12)

Address: 10224

```
property ain12_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@12)

Address: 10524

```
property ain12_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@12)

Address: 10824

```
property ain12_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@12)

Address: 11124

```
property ain12_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@12)

Address: 11424

```
property ain12_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@12)

Address: 11724

```
property ain12_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@12)

Address: 12024

```
property ain12_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN12_EF_INDEX (AIN#(0:149)_EF_INDEX@12)

Address: 9024

```
property ain12_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN12_EF_READ_A (AIN#(0:149)_EF_READ_A@12)

Address: 7024

```
property ain12_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN12_EF_READ_B (AIN#(0:149)_EF_READ_B@12)

Address: 7324

```
property ain12_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN12_EF_READ_C (AIN#(0:149)_EF_READ_C@12)

Address: 7624

```
property ain12_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN12_EF_READ_D (AIN#(0:149)_EF_READ_D@12)

Address: 7924

```
property ain12_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN12_NEGATIVE_CH (AIN#(0:13)_NEGATIVE_CH@12)

Address: 41012

```
property ain12_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN12_RANGE (AIN#(0:13)_RANGE@12)

Address: 40024

```
property ain12_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN12_RESOLUTION_INDEX (AIN#(0:13)_RESOLUTION_INDEX@12)

Address: 41512

```
property ain12_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN12_SETTLING_US (AIN#(0:13)_SETTLING_US@12)

Address: 42024

```
property ain13: float
```

Returns the voltage of the specified analog input.

Register: AIN13 (AIN#(0:13)@13)

Address: 26

```
property ain130_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@130)

Address: 9560

```
property ain130_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@130)

Address: 9860

```
property ain130_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@130)

Address: 10160

```
property ain130_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@130)

Address: 10460

```
property ain130_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@130)

Address: 10760

```
property ain130_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@130)

Address: 11060

```
property ain130_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@130)

Address: 11360

```
property ain130_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@130)

Address: 11660

```
property ain130_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@130)

Address: 11960

```
property ain130_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@130)

Address: 12260

```
property ain130_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN130_EF_INDEX (AIN#(0:149)_EF_INDEX@130)

Address: 9260

```
property ain130_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN130_EF_READ_A (AIN#(0:149)_EF_READ_A@130)

Address: 7260

```
property ain130_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN130_EF_READ_B (AIN#(0:149)_EF_READ_B@130)

Address: 7560

```
property ain130_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN130_EF_READ_C (AIN#(0:149)_EF_READ_C@130)

Address: 7860

```
property ain130_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN130_EF_READ_D (AIN#(0:149)_EF_READ_D@130)

Address: 8160

```
property ain131_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@131)

Address: 9562

```
property ain131_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@131)

Address: 9862

```
property ain131_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@131)

Address: 10162

```
property ain131_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@131)

Address: 10462

```
property ain131_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@131)

Address: 10762

```
property ain131_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@131)

Address: 11062

```
property ain131_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@131)

Address: 11362

```
property ain131_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@131)

Address: 11662

```
property ain131_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@131)

Address: 11962

```
property ain131_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@131)

Address: 12262

```
property ain131_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN131_EF_INDEX (AIN#(0:149)_EF_INDEX@131)

Address: 9262

```
property ain131_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN131_EF_READ_A (AIN#(0:149)_EF_READ_A@131)

Address: 7262

```
property ain131_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN131_EF_READ_B (AIN#(0:149)_EF_READ_B@131)

Address: 7562

```
property ain131_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN131_EF_READ_C (AIN#(0:149)_EF_READ_C@131)

Address: 7862

```
property ain131_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN131_EF_READ_D (AIN#(0:149)_EF_READ_D@131)

Address: 8162

```
property ain132_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@132)

Address: 9564

```
property ain132_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@132)

Address: 9864

```
property ain132_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@132)

Address: 10164

```
property ain132_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@132)

Address: 10464

```
property ain132_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@132)

Address: 10764

```
property ain132_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@132)

Address: 11064

```
property ain132_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@132)

Address: 11364

```
property ain132_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@132)

Address: 11664

```
property ain132_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@132)

Address: 11964

```
property ain132_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@132)

Address: 12264

```
property ain132_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN132_EF_INDEX (AIN#(0:149)_EF_INDEX@132)

Address: 9264

```
property ain132_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN132_EF_READ_A (AIN#(0:149)_EF_READ_A@132)

Address: 7264

```
property ain132_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN132_EF_READ_B (AIN#(0:149)_EF_READ_B@132)

Address: 7564

```
property ain132_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN132_EF_READ_C (AIN#(0:149)_EF_READ_C@132)

Address: 7864

```
property ain132_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN132_EF_READ_D (AIN#(0:149)_EF_READ_D@132)

Address: 8164

```
property ain133_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@133)

Address: 9566

```
property ain133_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@133)

Address: 9866

```
property ain133_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@133)

Address: 10166

```
property ain133_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@133)

Address: 10466

```
property ain133_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@133)

Address: 10766

```
property ain133_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@133)

Address: 11066

```
property ain133_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@133)

Address: 11366

```
property ain133_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@133)

Address: 11666

```
property ain133_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@133)

Address: 11966

```
property ain133_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@133)

Address: 12266

```
property ain133_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN133_EF_INDEX (AIN#(0:149)_EF_INDEX@133)

Address: 9266

```
property ain133_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN133_EF_READ_A (AIN#(0:149)_EF_READ_A@133)

Address: 7266

```
property ain133_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN133_EF_READ_B (AIN#(0:149)_EF_READ_B@133)

Address: 7566

```
property ain133_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN133_EF_READ_C (AIN#(0:149)_EF_READ_C@133)

Address: 7866

```
property ain133_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN133_EF_READ_D (AIN#(0:149)_EF_READ_D@133)

Address: 8166

```
property ain134_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@134)

Address: 9568

```
property ain134_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@134)

Address: 9868

```
property ain134_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@134)

Address: 10168

```
property ain134_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@134)

Address: 10468

```
property ain134_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@134)

Address: 10768

```
property ain134_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@134)

Address: 11068

```
property ain134_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@134)

Address: 11368

```
property ain134_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@134)

Address: 11668

```
property ain134_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@134)

Address: 11968

```
property ain134_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@134)

Address: 12268

```
property ain134_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN134_EF_INDEX (AIN#(0:149)_EF_INDEX@134)

Address: 9268

```
property ain134_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN134_EF_READ_A (AIN#(0:149)_EF_READ_A@134)

Address: 7268

```
property ain134_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN134_EF_READ_B (AIN#(0:149)_EF_READ_B@134)

Address: 7568

```
property ain134_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN134_EF_READ_C (AIN#(0:149)_EF_READ_C@134)

Address: 7868

```
property ain134_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN134_EF_READ_D (AIN#(0:149)_EF_READ_D@134)

Address: 8168

```
property ain135_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@135)

Address: 9570

```
property ain135_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@135)

Address: 9870

```
property ain135_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@135)

Address: 10170

```
property ain135_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@135)

Address: 10470

```
property ain135_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@135)

Address: 10770

```
property ain135_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@135)

Address: 11070

```
property ain135_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@135)

Address: 11370

```
property ain135_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@135)

Address: 11670

```
property ain135_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@135)

Address: 11970

```
property ain135_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@135)

Address: 12270

```
property ain135_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN135_EF_INDEX (AIN#(0:149)_EF_INDEX@135)

Address: 9270

```
property ain135_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN135_EF_READ_A (AIN#(0:149)_EF_READ_A@135)

Address: 7270

```
property ain135_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN135_EF_READ_B (AIN#(0:149)_EF_READ_B@135)

Address: 7570

```
property ain135_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN135_EF_READ_C (AIN#(0:149)_EF_READ_C@135)

Address: 7870

```
property ain135_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN135_EF_READ_D (AIN#(0:149)_EF_READ_D@135)

Address: 8170

```
property ain136_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@136)

Address: 9572

```
property ain136_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@136)

Address: 9872

```
property ain136_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@136)

Address: 10172

```
property ain136_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@136)

Address: 10472

```
property ain136_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@136)

Address: 10772

```
property ain136_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@136)

Address: 11072

```
property ain136_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@136)

Address: 11372

```
property ain136_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@136)

Address: 11672

```
property ain136_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@136)

Address: 11972

```
property ain136_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@136)

Address: 12272

```
property ain136_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN136_EF_INDEX (AIN#(0:149)_EF_INDEX@136)

Address: 9272

```
property ain136_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN136_EF_READ_A (AIN#(0:149)_EF_READ_A@136)

Address: 7272

```
property ain136_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN136_EF_READ_B (AIN#(0:149)_EF_READ_B@136)

Address: 7572

```
property ain136_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN136_EF_READ_C (AIN#(0:149)_EF_READ_C@136)

Address: 7872

```
property ain136_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN136_EF_READ_D (AIN#(0:149)_EF_READ_D@136)

Address: 8172

```
property ain137_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@137)

Address: 9574

```
property ain137_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@137)

Address: 9874

```
property ain137_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@137)

Address: 10174

```
property ain137_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@137)

Address: 10474

```
property ain137_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@137)

Address: 10774

```
property ain137_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@137)

Address: 11074

```
property ain137_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@137)

Address: 11374

```
property ain137_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@137)

Address: 11674

```
property ain137_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@137)

Address: 11974

```
property ain137_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@137)

Address: 12274

```
property ain137_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN137_EF_INDEX (AIN#(0:149)_EF_INDEX@137)

Address: 9274

```
property ain137_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN137_EF_READ_A (AIN#(0:149)_EF_READ_A@137)

Address: 7274

```
property ain137_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN137_EF_READ_B (AIN#(0:149)_EF_READ_B@137)

Address: 7574

```
property ain137_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN137_EF_READ_C (AIN#(0:149)_EF_READ_C@137)

Address: 7874

```
property ain137_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN137_EF_READ_D (AIN#(0:149)_EF_READ_D@137)

Address: 8174

```
property ain138_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@138)

Address: 9576

```
property ain138_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@138)

Address: 9876

```
property ain138_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@138)

Address: 10176

```
property ain138_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@138)

Address: 10476

```
property ain138_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@138)

Address: 10776

```
property ain138_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@138)

Address: 11076

```
property ain138_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@138)

Address: 11376

```
property ain138_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@138)

Address: 11676

```
property ain138_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@138)

Address: 11976

```
property ain138_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@138)

Address: 12276

```
property ain138_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN138_EF_INDEX (AIN#(0:149)_EF_INDEX@138)

Address: 9276

```
property ain138_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN138_EF_READ_A (AIN#(0:149)_EF_READ_A@138)

Address: 7276

```
property ain138_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN138_EF_READ_B (AIN#(0:149)_EF_READ_B@138)

Address: 7576

```
property ain138_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN138_EF_READ_C (AIN#(0:149)_EF_READ_C@138)

Address: 7876

```
property ain138_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN138_EF_READ_D (AIN#(0:149)_EF_READ_D@138)

Address: 8176

```
property ain139_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@139)

Address: 9578

```
property ain139_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@139)

Address: 9878

```
property ain139_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@139)

Address: 10178

```
property ain139_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@139)

Address: 10478

```
property ain139_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@139)

Address: 10778

```
property ain139_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@139)

Address: 11078

```
property ain139_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@139)

Address: 11378

```
property ain139_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@139)

Address: 11678

```
property ain139_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@139)

Address: 11978

```
property ain139_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@139)

Address: 12278

```
property ain139_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN139_EF_INDEX (AIN#(0:149)_EF_INDEX@139)

Address: 9278

```
property ain139_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN139_EF_READ_A (AIN#(0:149)_EF_READ_A@139)

Address: 7278

```
property ain139_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN139_EF_READ_B (AIN#(0:149)_EF_READ_B@139)

Address: 7578

```
property ain139_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN139_EF_READ_C (AIN#(0:149)_EF_READ_C@139)

Address: 7878

```
property ain139_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN139_EF_READ_D (AIN#(0:149)_EF_READ_D@139)

Address: 8178

```
property ain13_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN13_BINARY (AIN#(0:13)_BINARY@13)

Address: 50026

```
property ain13_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@13)

Address: 9326

```
property ain13_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@13)

Address: 9626

```
property ain13_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@13)

Address: 9926

```
property ain13_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@13)

Address: 10226

```
property ain13_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@13)

Address: 10526

```
property ain13_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@13)

Address: 10826

```
property ain13_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@13)

Address: 11126

```
property ain13_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@13)

Address: 11426

```
property ain13_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@13)

Address: 11726

```
property ain13_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@13)

Address: 12026

```
property ain13_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN13_EF_INDEX (AIN#(0:149)_EF_INDEX@13)

Address: 9026

```
property ain13_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN13_EF_READ_A (AIN#(0:149)_EF_READ_A@13)

Address: 7026

```
property ain13_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN13_EF_READ_B (AIN#(0:149)_EF_READ_B@13)

Address: 7326

```
property ain13_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN13_EF_READ_C (AIN#(0:149)_EF_READ_C@13)

Address: 7626

```
property ain13_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN13_EF_READ_D (AIN#(0:149)_EF_READ_D@13)

Address: 7926

```
property ain13_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN13_NEGATIVE_CH (AIN#(0:13)_NEGATIVE_CH@13)

Address: 41013

```
property ain13_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN13_RANGE (AIN#(0:13)_RANGE@13)

Address: 40026

```
property ain13_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN13_RESOLUTION_INDEX (AIN#(0:13)_RESOLUTION_INDEX@13)

Address: 41513

```
property ain13_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN13_SETTLING_US (AIN#(0:13)_SETTLING_US@13)

Address: 42026

```
property ain140_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@140)

Address: 9580

```
property ain140_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@140)

Address: 9880

```
property ain140_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@140)

Address: 10180

```
property ain140_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@140)

Address: 10480

```
property ain140_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@140)

Address: 10780

```
property ain140_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@140)

Address: 11080

```
property ain140_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@140)

Address: 11380

```
property ain140_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@140)

Address: 11680

```
property ain140_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@140)

Address: 11980

```
property ain140_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@140)

Address: 12280

```
property ain140_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN140_EF_INDEX (AIN#(0:149)_EF_INDEX@140)

Address: 9280

```
property ain140_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN140_EF_READ_A (AIN#(0:149)_EF_READ_A@140)

Address: 7280

```
property ain140_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN140_EF_READ_B (AIN#(0:149)_EF_READ_B@140)

Address: 7580

```
property ain140_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN140_EF_READ_C (AIN#(0:149)_EF_READ_C@140)

Address: 7880

```
property ain140_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN140_EF_READ_D (AIN#(0:149)_EF_READ_D@140)

Address: 8180

```
property ain141_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@141)

Address: 9582

```
property ain141_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@141)

Address: 9882

```
property ain141_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@141)

Address: 10182

```
property ain141_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@141)

Address: 10482

```
property ain141_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@141)

Address: 10782

```
property ain141_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@141)

Address: 11082

```
property ain141_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@141)

Address: 11382

```
property ain141_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@141)

Address: 11682

```
property ain141_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@141)

Address: 11982

```
property ain141_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@141)

Address: 12282

```
property ain141_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN141_EF_INDEX (AIN#(0:149)_EF_INDEX@141)

Address: 9282

```
property ain141_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN141_EF_READ_A (AIN#(0:149)_EF_READ_A@141)

Address: 7282

```
property ain141_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN141_EF_READ_B (AIN#(0:149)_EF_READ_B@141)

Address: 7582

```
property ain141_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN141_EF_READ_C (AIN#(0:149)_EF_READ_C@141)

Address: 7882

```
property ain141_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN141_EF_READ_D (AIN#(0:149)_EF_READ_D@141)

Address: 8182

```
property ain142_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@142)

Address: 9584

```
property ain142_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@142)

Address: 9884

```
property ain142_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@142)

Address: 10184

```
property ain142_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@142)

Address: 10484

```
property ain142_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@142)

Address: 10784

```
property ain142_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@142)

Address: 11084

```
property ain142_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@142)

Address: 11384

```
property ain142_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@142)

Address: 11684

```
property ain142_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@142)

Address: 11984

```
property ain142_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@142)

Address: 12284

```
property ain142_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN142_EF_INDEX (AIN#(0:149)_EF_INDEX@142)

Address: 9284

```
property ain142_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN142_EF_READ_A (AIN#(0:149)_EF_READ_A@142)

Address: 7284

```
property ain142_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN142_EF_READ_B (AIN#(0:149)_EF_READ_B@142)

Address: 7584

```
property ain142_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN142_EF_READ_C (AIN#(0:149)_EF_READ_C@142)

Address: 7884

```
property ain142_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN142_EF_READ_D (AIN#(0:149)_EF_READ_D@142)

Address: 8184

```
property ain143_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@143)

Address: 9586

```
property ain143_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@143)

Address: 9886

```
property ain143_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@143)

Address: 10186

```
property ain143_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@143)

Address: 10486

```
property ain143_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@143)

Address: 10786

```
property ain143_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@143)

Address: 11086

```
property ain143_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@143)

Address: 11386

```
property ain143_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@143)

Address: 11686

```
property ain143_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@143)

Address: 11986

```
property ain143_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@143)

Address: 12286

```
property ain143_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN143_EF_INDEX (AIN#(0:149)_EF_INDEX@143)

Address: 9286

```
property ain143_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN143_EF_READ_A (AIN#(0:149)_EF_READ_A@143)

Address: 7286

```
property ain143_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN143_EF_READ_B (AIN#(0:149)_EF_READ_B@143)

Address: 7586

```
property ain143_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN143_EF_READ_C (AIN#(0:149)_EF_READ_C@143)

Address: 7886

```
property ain143_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN143_EF_READ_D (AIN#(0:149)_EF_READ_D@143)

Address: 8186

```
property ain144_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@144)

Address: 9588

```
property ain144_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@144)

Address: 9888

```
property ain144_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@144)

Address: 10188

```
property ain144_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@144)

Address: 10488

```
property ain144_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@144)

Address: 10788

```
property ain144_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@144)

Address: 11088

```
property ain144_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@144)

Address: 11388

```
property ain144_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@144)

Address: 11688

```
property ain144_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@144)

Address: 11988

```
property ain144_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@144)

Address: 12288

```
property ain144_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN144_EF_INDEX (AIN#(0:149)_EF_INDEX@144)

Address: 9288

```
property ain144_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN144_EF_READ_A (AIN#(0:149)_EF_READ_A@144)

Address: 7288

```
property ain144_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN144_EF_READ_B (AIN#(0:149)_EF_READ_B@144)

Address: 7588

```
property ain144_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN144_EF_READ_C (AIN#(0:149)_EF_READ_C@144)

Address: 7888

```
property ain144_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN144_EF_READ_D (AIN#(0:149)_EF_READ_D@144)

Address: 8188

```
property ain145_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@145)

Address: 9590

```
property ain145_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@145)

Address: 9890

```
property ain145_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@145)

Address: 10190

```
property ain145_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@145)

Address: 10490

```
property ain145_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@145)

Address: 10790

```
property ain145_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@145)

Address: 11090

```
property ain145_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@145)

Address: 11390

```
property ain145_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@145)

Address: 11690

```
property ain145_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@145)

Address: 11990

```
property ain145_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@145)

Address: 12290

```
property ain145_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN145_EF_INDEX (AIN#(0:149)_EF_INDEX@145)

Address: 9290

```
property ain145_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN145_EF_READ_A (AIN#(0:149)_EF_READ_A@145)

Address: 7290

```
property ain145_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN145_EF_READ_B (AIN#(0:149)_EF_READ_B@145)

Address: 7590

```
property ain145_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN145_EF_READ_C (AIN#(0:149)_EF_READ_C@145)

Address: 7890

```
property ain145_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN145_EF_READ_D (AIN#(0:149)_EF_READ_D@145)

Address: 8190

```
property ain146_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@146)

Address: 9592

```
property ain146_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@146)

Address: 9892

```
property ain146_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@146)

Address: 10192

```
property ain146_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@146)

Address: 10492

```
property ain146_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@146)

Address: 10792

```
property ain146_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@146)

Address: 11092

```
property ain146_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@146)

Address: 11392

```
property ain146_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@146)

Address: 11692

```
property ain146_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@146)

Address: 11992

```
property ain146_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@146)

Address: 12292

```
property ain146_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN146_EF_INDEX (AIN#(0:149)_EF_INDEX@146)

Address: 9292

```
property ain146_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN146_EF_READ_A (AIN#(0:149)_EF_READ_A@146)

Address: 7292

```
property ain146_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN146_EF_READ_B (AIN#(0:149)_EF_READ_B@146)

Address: 7592

```
property ain146_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN146_EF_READ_C (AIN#(0:149)_EF_READ_C@146)

Address: 7892

```
property ain146_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN146_EF_READ_D (AIN#(0:149)_EF_READ_D@146)

Address: 8192

```
property ain147_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@147)

Address: 9594

```
property ain147_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@147)

Address: 9894

```
property ain147_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@147)

Address: 10194

```
property ain147_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@147)

Address: 10494

```
property ain147_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@147)

Address: 10794

```
property ain147_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@147)

Address: 11094

```
property ain147_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@147)

Address: 11394

```
property ain147_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@147)

Address: 11694

```
property ain147_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@147)

Address: 11994

```
property ain147_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@147)

Address: 12294

```
property ain147_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN147_EF_INDEX (AIN#(0:149)_EF_INDEX@147)

Address: 9294

```
property ain147_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN147_EF_READ_A (AIN#(0:149)_EF_READ_A@147)

Address: 7294

```
property ain147_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN147_EF_READ_B (AIN#(0:149)_EF_READ_B@147)

Address: 7594

```
property ain147_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN147_EF_READ_C (AIN#(0:149)_EF_READ_C@147)

Address: 7894

```
property ain147_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN147_EF_READ_D (AIN#(0:149)_EF_READ_D@147)

Address: 8194

```
property ain148_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@148)

Address: 9596

```
property ain148_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@148)

Address: 9896

```
property ain148_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@148)

Address: 10196

```
property ain148_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@148)

Address: 10496

```
property ain148_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@148)

Address: 10796

```
property ain148_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@148)

Address: 11096

```
property ain148_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@148)

Address: 11396

```
property ain148_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@148)

Address: 11696

```
property ain148_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@148)

Address: 11996

```
property ain148_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@148)

Address: 12296

```
property ain148_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN148_EF_INDEX (AIN#(0:149)_EF_INDEX@148)

Address: 9296

```
property ain148_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN148_EF_READ_A (AIN#(0:149)_EF_READ_A@148)

Address: 7296

```
property ain148_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN148_EF_READ_B (AIN#(0:149)_EF_READ_B@148)

Address: 7596

```
property ain148_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN148_EF_READ_C (AIN#(0:149)_EF_READ_C@148)

Address: 7896

```
property ain148_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN148_EF_READ_D (AIN#(0:149)_EF_READ_D@148)

Address: 8196

```
property ain149_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@149)

Address: 9598

```
property ain149_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@149)

Address: 9898

```
property ain149_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@149)

Address: 10198

```
property ain149_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@149)

Address: 10498

```
property ain149_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@149)

Address: 10798

```
property ain149_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@149)

Address: 11098

```
property ain149_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@149)

Address: 11398

```
property ain149_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@149)

Address: 11698

```
property ain149_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@149)

Address: 11998

```
property ain149_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@149)

Address: 12298

```
property ain149_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN149_EF_INDEX (AIN#(0:149)_EF_INDEX@149)

Address: 9298

```
property ain149_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN149_EF_READ_A (AIN#(0:149)_EF_READ_A@149)

Address: 7298

```
property ain149_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN149_EF_READ_B (AIN#(0:149)_EF_READ_B@149)

Address: 7598

```
property ain149_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN149_EF_READ_C (AIN#(0:149)_EF_READ_C@149)

Address: 7898

```
property ain149_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN149_EF_READ_D (AIN#(0:149)_EF_READ_D@149)

Address: 8198

```
property ain14_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@14)

Address: 9328

```
property ain14_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@14)

Address: 9628

```
property ain14_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@14)

Address: 9928

```
property ain14_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@14)

Address: 10228

```
property ain14_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@14)

Address: 10528

```
property ain14_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@14)

Address: 10828

```
property ain14_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@14)

Address: 11128

```
property ain14_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@14)

Address: 11428

```
property ain14_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@14)

Address: 11728

```
property ain14_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@14)

Address: 12028

```
property ain14_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN14_EF_INDEX (AIN#(0:149)_EF_INDEX@14)

Address: 9028

```
property ain14_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN14_EF_READ_A (AIN#(0:149)_EF_READ_A@14)

Address: 7028

```
property ain14_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN14_EF_READ_B (AIN#(0:149)_EF_READ_B@14)

Address: 7328

```
property ain14_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN14_EF_READ_C (AIN#(0:149)_EF_READ_C@14)

Address: 7628

```
property ain14_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN14_EF_READ_D (AIN#(0:149)_EF_READ_D@14)

Address: 7928

```
property ain15_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@15)

Address: 9330

```
property ain15_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@15)

Address: 9630

```
property ain15_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@15)

Address: 9930

```
property ain15_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@15)

Address: 10230

```
property ain15_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@15)

Address: 10530

```
property ain15_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@15)

Address: 10830

```
property ain15_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@15)

Address: 11130

```
property ain15_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@15)

Address: 11430

```
property ain15_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@15)

Address: 11730

```
property ain15_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@15)

Address: 12030

```
property ain15_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN15_EF_INDEX (AIN#(0:149)_EF_INDEX@15)

Address: 9030

```
property ain15_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN15_EF_READ_A (AIN#(0:149)_EF_READ_A@15)

Address: 7030

```
property ain15_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN15_EF_READ_B (AIN#(0:149)_EF_READ_B@15)

Address: 7330

```
property ain15_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN15_EF_READ_C (AIN#(0:149)_EF_READ_C@15)

Address: 7630

```
property ain15_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN15_EF_READ_D (AIN#(0:149)_EF_READ_D@15)

Address: 7930

```
property ain16_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@16)

Address: 9332

```
property ain16_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@16)

Address: 9632

```
property ain16_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@16)

Address: 9932

```
property ain16_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@16)

Address: 10232

```
property ain16_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@16)

Address: 10532

```
property ain16_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@16)

Address: 10832

```
property ain16_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@16)

Address: 11132

```
property ain16_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@16)

Address: 11432

```
property ain16_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@16)

Address: 11732

```
property ain16_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@16)

Address: 12032

```
property ain16_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN16_EF_INDEX (AIN#(0:149)_EF_INDEX@16)

Address: 9032

```
property ain16_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN16_EF_READ_A (AIN#(0:149)_EF_READ_A@16)

Address: 7032

```
property ain16_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN16_EF_READ_B (AIN#(0:149)_EF_READ_B@16)

Address: 7332

```
property ain16_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN16_EF_READ_C (AIN#(0:149)_EF_READ_C@16)

Address: 7632

```
property ain16_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN16_EF_READ_D (AIN#(0:149)_EF_READ_D@16)

Address: 7932

```
property ain17_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@17)

Address: 9334

```
property ain17_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@17)

Address: 9634

```
property ain17_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@17)

Address: 9934

```
property ain17_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@17)

Address: 10234

```
property ain17_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@17)

Address: 10534

```
property ain17_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@17)

Address: 10834

```
property ain17_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@17)

Address: 11134

```
property ain17_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@17)

Address: 11434

```
property ain17_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@17)

Address: 11734

```
property ain17_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@17)

Address: 12034

```
property ain17_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN17_EF_INDEX (AIN#(0:149)_EF_INDEX@17)

Address: 9034

```
property ain17_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN17_EF_READ_A (AIN#(0:149)_EF_READ_A@17)

Address: 7034

```
property ain17_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN17_EF_READ_B (AIN#(0:149)_EF_READ_B@17)

Address: 7334

```
property ain17_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN17_EF_READ_C (AIN#(0:149)_EF_READ_C@17)

Address: 7634

```
property ain17_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN17_EF_READ_D (AIN#(0:149)_EF_READ_D@17)

Address: 7934

```
property ain18_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@18)

Address: 9336

```
property ain18_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@18)

Address: 9636

```
property ain18_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@18)

Address: 9936

```
property ain18_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@18)

Address: 10236

```
property ain18_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@18)

Address: 10536

```
property ain18_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@18)

Address: 10836

```
property ain18_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@18)

Address: 11136

```
property ain18_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@18)

Address: 11436

```
property ain18_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@18)

Address: 11736

```
property ain18_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@18)

Address: 12036

```
property ain18_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN18_EF_INDEX (AIN#(0:149)_EF_INDEX@18)

Address: 9036

```
property ain18_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN18_EF_READ_A (AIN#(0:149)_EF_READ_A@18)

Address: 7036

```
property ain18_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN18_EF_READ_B (AIN#(0:149)_EF_READ_B@18)

Address: 7336

```
property ain18_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN18_EF_READ_C (AIN#(0:149)_EF_READ_C@18)

Address: 7636

```
property ain18_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN18_EF_READ_D (AIN#(0:149)_EF_READ_D@18)

Address: 7936

```
property ain19_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@19)

Address: 9338

```
property ain19_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@19)

Address: 9638

```
property ain19_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@19)

Address: 9938

```
property ain19_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@19)

Address: 10238

```
property ain19_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@19)

Address: 10538

```
property ain19_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@19)

Address: 10838

```
property ain19_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@19)

Address: 11138

```
property ain19_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@19)

Address: 11438

```
property ain19_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@19)

Address: 11738

```
property ain19_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@19)

Address: 12038

```
property ain19_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN19_EF_INDEX (AIN#(0:149)_EF_INDEX@19)

Address: 9038

```
property ain19_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN19_EF_READ_A (AIN#(0:149)_EF_READ_A@19)

Address: 7038

```
property ain19_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN19_EF_READ_B (AIN#(0:149)_EF_READ_B@19)

Address: 7338

```
property ain19_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN19_EF_READ_C (AIN#(0:149)_EF_READ_C@19)

Address: 7638

```
property ain19_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN19_EF_READ_D (AIN#(0:149)_EF_READ_D@19)

Address: 7938

```
property ain1_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN1_BINARY (AIN#(0:13)_BINARY@1)

Address: 50002

```
property ain1_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@1)

Address: 9302

```
property ain1_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@1)

Address: 9602

```
property ain1_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@1)

Address: 9902

```
property ain1_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@1)

Address: 10202

```
property ain1_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@1)

Address: 10502

```
property ain1_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@1)

Address: 10802

```
property ain1_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@1)

Address: 11102

```
property ain1_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@1)

Address: 11402

```
property ain1_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@1)

Address: 11702

```
property ain1_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@1)

Address: 12002

```
property ain1_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN1_EF_INDEX (AIN#(0:149)_EF_INDEX@1)

Address: 9002

```
property ain1_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN1_EF_READ_A (AIN#(0:149)_EF_READ_A@1)

Address: 7002

```
property ain1_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN1_EF_READ_B (AIN#(0:149)_EF_READ_B@1)

Address: 7302

```
property ain1_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN1_EF_READ_C (AIN#(0:149)_EF_READ_C@1)

Address: 7602

```
property ain1_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN1_EF_READ_D (AIN#(0:149)_EF_READ_D@1)

Address: 7902

```
property ain1_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN1_NEGATIVE_CH (AIN#(0:13)_NEGATIVE_CH@1)

Address: 41001

```
property ain1_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN1_RANGE (AIN#(0:13)_RANGE@1)

Address: 40002

```
property ain1_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN1_RESOLUTION_INDEX (AIN#(0:13)_RESOLUTION_INDEX@1)

Address: 41501

```
property ain1_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN1_SETTLING_US (AIN#(0:13)_SETTLING_US@1)

Address: 42002

```
property ain2: float
```

Returns the voltage of the specified analog input.

Register: AIN2 (AIN#(0:13)@2)

Address: 4

```
property ain20_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@20)

Address: 9340

```
property ain20_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@20)

Address: 9640

```
property ain20_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@20)

Address: 9940

```
property ain20_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@20)

Address: 10240

```
property ain20_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@20)

Address: 10540

```
property ain20_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@20)

Address: 10840

```
property ain20_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@20)

Address: 11140

```
property ain20_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@20)

Address: 11440

```
property ain20_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@20)

Address: 11740

```
property ain20_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@20)

Address: 12040

```
property ain20_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T;

25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN20_EF_INDEX (AIN#(0:149)_EF_INDEX@20)

Address: 9040

```
property ain20_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN20_EF_READ_A (AIN#(0:149)_EF_READ_A@20)

Address: 7040

```
property ain20_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN20_EF_READ_B (AIN#(0:149)_EF_READ_B@20)

Address: 7340

```
property ain20_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN20_EF_READ_C (AIN#(0:149)_EF_READ_C@20)

Address: 7640

```
property ain20_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN20_EF_READ_D (AIN#(0:149)_EF_READ_D@20)

Address: 7940

```
property ain21_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@21)

Address: 9342

```
property ain21_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@21)

Address: 9642

```
property ain21_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@21)

Address: 9942

```
property ain21_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@21)

Address: 10242

```
property ain21_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@21)

Address: 10542

```
property ain21_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@21)

Address: 10842

```
property ain21_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@21)

Address: 11142

```
property ain21_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@21)

Address: 11442

```
property ain21_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@21)

Address: 11742

```
property ain21_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@21)

Address: 12042

```
property ain21_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN21_EF_INDEX (AIN#(0:149)_EF_INDEX@21)

Address: 9042

```
property ain21_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN21_EF_READ_A (AIN#(0:149)_EF_READ_A@21)

Address: 7042

```
property ain21_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN21_EF_READ_B (AIN#(0:149)_EF_READ_B@21)

Address: 7342

```
property ain21_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN21_EF_READ_C (AIN#(0:149)_EF_READ_C@21)

Address: 7642

```
property ain21_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN21_EF_READ_D (AIN#(0:149)_EF_READ_D@21)

Address: 7942

```
property ain22_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@22)

Address: 9344

```
property ain22_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@22)

Address: 9644

```
property ain22_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@22)

Address: 9944

```
property ain22_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@22)

Address: 10244

```
property ain22_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@22)

Address: 10544

```
property ain22_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@22)

Address: 10844

```
property ain22_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@22)

Address: 11144

```
property ain22_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@22)

Address: 11444

```
property ain22_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@22)

Address: 11744

```
property ain22_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@22)

Address: 12044

```
property ain22_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN22_EF_INDEX (AIN#(0:149)_EF_INDEX@22)

Address: 9044

```
property ain22_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN22_EF_READ_A (AIN#(0:149)_EF_READ_A@22)

Address: 7044

```
property ain22_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN22_EF_READ_B (AIN#(0:149)_EF_READ_B@22)

Address: 7344

```
property ain22_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN22_EF_READ_C (AIN#(0:149)_EF_READ_C@22)

Address: 7644

```
property ain22_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN22_EF_READ_D (AIN#(0:149)_EF_READ_D@22)

Address: 7944

```
property ain23_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@23)

Address: 9346

```
property ain23_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@23)

Address: 9646

```
property ain23_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@23)

Address: 9946

```
property ain23_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@23)

Address: 10246

```
property ain23_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@23)

Address: 10546

```
property ain23_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@23)

Address: 10846

```
property ain23_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@23)

Address: 11146

```
property ain23_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@23)

Address: 11446

```
property ain23_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@23)

Address: 11746

```
property ain23_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@23)

Address: 12046

```
property ain23_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN23_EF_INDEX (AIN#(0:149)_EF_INDEX@23)

Address: 9046

```
property ain23_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN23_EF_READ_A (AIN#(0:149)_EF_READ_A@23)

Address: 7046

```
property ain23_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN23_EF_READ_B (AIN#(0:149)_EF_READ_B@23)

Address: 7346

```
property ain23_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN23_EF_READ_C (AIN#(0:149)_EF_READ_C@23)

Address: 7646

```
property ain23_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN23_EF_READ_D (AIN#(0:149)_EF_READ_D@23)

Address: 7946

```
property ain24_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@24)

Address: 9348

```
property ain24_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@24)

Address: 9648

```
property ain24_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@24)

Address: 9948

```
property ain24_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@24)

Address: 10248

```
property ain24_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@24)

Address: 10548

```
property ain24_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@24)

Address: 10848

```
property ain24_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@24)

Address: 11148

```
property ain24_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@24)

Address: 11448

```
property ain24_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@24)

Address: 11748

```
property ain24_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@24)

Address: 12048

```
property ain24_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN24_EF_INDEX (AIN#(0:149)_EF_INDEX@24)

Address: 9048

```
property ain24_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN24_EF_READ_A (AIN#(0:149)_EF_READ_A@24)

Address: 7048

```
property ain24_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN24_EF_READ_B (AIN#(0:149)_EF_READ_B@24)

Address: 7348

```
property ain24_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN24_EF_READ_C (AIN#(0:149)_EF_READ_C@24)

Address: 7648

```
property ain24_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN24_EF_READ_D (AIN#(0:149)_EF_READ_D@24)

Address: 7948

```
property ain25_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@25)

Address: 9350

```
property ain25_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@25)

Address: 9650

```
property ain25_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@25)

Address: 9950

```
property ain25_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@25)

Address: 10250

```
property ain25_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@25)

Address: 10550

```
property ain25_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@25)

Address: 10850

```
property ain25_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@25)

Address: 11150

```
property ain25_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@25)

Address: 11450

```
property ain25_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@25)

Address: 11750

```
property ain25_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@25)

Address: 12050

```
property ain25_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN25_EF_INDEX (AIN#(0:149)_EF_INDEX@25)

Address: 9050

```
property ain25_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN25_EF_READ_A (AIN#(0:149)_EF_READ_A@25)

Address: 7050

```
property ain25_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN25_EF_READ_B (AIN#(0:149)_EF_READ_B@25)

Address: 7350

```
property ain25_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN25_EF_READ_C (AIN#(0:149)_EF_READ_C@25)

Address: 7650

```
property ain25_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN25_EF_READ_D (AIN#(0:149)_EF_READ_D@25)

Address: 7950

```
property ain26_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@26)

Address: 9352

```
property ain26_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@26)

Address: 9652

```
property ain26_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@26)

Address: 9952

```
property ain26_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@26)

Address: 10252

```
property ain26_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@26)

Address: 10552

```
property ain26_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@26)

Address: 10852

```
property ain26_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@26)

Address: 11152

```
property ain26_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@26)

Address: 11452

```
property ain26_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@26)

Address: 11752

```
property ain26_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@26)

Address: 12052

```
property ain26_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN26_EF_INDEX (AIN#(0:149)_EF_INDEX@26)

Address: 9052

```
property ain26_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN26_EF_READ_A (AIN#(0:149)_EF_READ_A@26)

Address: 7052

```
property ain26_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN26_EF_READ_B (AIN#(0:149)_EF_READ_B@26)

Address: 7352

```
property ain26_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN26_EF_READ_C (AIN#(0:149)_EF_READ_C@26)

Address: 7652

```
property ain26_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN26_EF_READ_D (AIN#(0:149)_EF_READ_D@26)

Address: 7952

```
property ain27_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@27)

Address: 9354

```
property ain27_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@27)

Address: 9654

```
property ain27_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@27)

Address: 9954

```
property ain27_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@27)

Address: 10254

```
property ain27_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@27)

Address: 10554

```
property ain27_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@27)

Address: 10854

```
property ain27_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@27)

Address: 11154

```
property ain27_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@27)

Address: 11454

```
property ain27_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@27)

Address: 11754

```
property ain27_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@27)

Address: 12054

```
property ain27_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN27_EF_INDEX (AIN#(0:149)_EF_INDEX@27)

Address: 9054

```
property ain27_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN27_EF_READ_A (AIN#(0:149)_EF_READ_A@27)

Address: 7054

```
property ain27_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN27_EF_READ_B (AIN#(0:149)_EF_READ_B@27)

Address: 7354

```
property ain27_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN27_EF_READ_C (AIN#(0:149)_EF_READ_C@27)

Address: 7654

```
property ain27_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN27_EF_READ_D (AIN#(0:149)_EF_READ_D@27)

Address: 7954

```
property ain28_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@28)

Address: 9356

```
property ain28_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@28)

Address: 9656

```
property ain28_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@28)

Address: 9956

```
property ain28_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@28)

Address: 10256

```
property ain28_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@28)

Address: 10556

```
property ain28_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@28)

Address: 10856

```
property ain28_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@28)

Address: 11156

```
property ain28_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@28)

Address: 11456

```
property ain28_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@28)

Address: 11756

```
property ain28_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@28)

Address: 12056

```
property ain28_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN28_EF_INDEX (AIN#(0:149)_EF_INDEX@28)

Address: 9056

```
property ain28_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN28_EF_READ_A (AIN#(0:149)_EF_READ_A@28)

Address: 7056

```
property ain28_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN28_EF_READ_B (AIN#(0:149)_EF_READ_B@28)

Address: 7356

```
property ain28_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN28_EF_READ_C (AIN#(0:149)_EF_READ_C@28)

Address: 7656

```
property ain28_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN28_EF_READ_D (AIN#(0:149)_EF_READ_D@28)

Address: 7956

```
property ain29_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@29)

Address: 9358

```
property ain29_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@29)

Address: 9658

```
property ain29_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@29)

Address: 9958

```
property ain29_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@29)

Address: 10258

```
property ain29_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@29)

Address: 10558

```
property ain29_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@29)

Address: 10858

```
property ain29_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@29)

Address: 11158

```
property ain29_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@29)

Address: 11458

```
property ain29_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@29)

Address: 11758

```
property ain29_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@29)

Address: 12058

```
property ain29_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN29_EF_INDEX (AIN#(0:149)_EF_INDEX@29)

Address: 9058

```
property ain29_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN29_EF_READ_A (AIN#(0:149)_EF_READ_A@29)

Address: 7058

```
property ain29_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN29_EF_READ_B (AIN#(0:149)_EF_READ_B@29)

Address: 7358

```
property ain29_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN29_EF_READ_C (AIN#(0:149)_EF_READ_C@29)

Address: 7658

```
property ain29_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN29_EF_READ_D (AIN#(0:149)_EF_READ_D@29)

Address: 7958

```
property ain2_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN2_BINARY (AIN#(0:13)_BINARY@2)

Address: 50004

```
property ain2_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@2)

Address: 9304

```
property ain2_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@2)

Address: 9604

```
property ain2_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@2)

Address: 9904

```
property ain2_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@2)

Address: 10204

```
property ain2_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@2)

Address: 10504

```
property ain2_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@2)

Address: 10804

```
property ain2_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@2)

Address: 11104

```
property ain2_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@2)

Address: 11404

```
property ain2_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@2)

Address: 11704

```
property ain2_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@2)

Address: 12004

```
property ain2_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN2_EF_INDEX (AIN#(0:149)_EF_INDEX@2)

Address: 9004

```
property ain2_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN2_EF_READ_A (AIN#(0:149)_EF_READ_A@2)

Address: 7004

```
property ain2_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN2_EF_READ_B (AIN#(0:149)_EF_READ_B@2)

Address: 7304

```
property ain2_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN2_EF_READ_C (AIN#(0:149)_EF_READ_C@2)

Address: 7604

```
property ain2_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN2_EF_READ_D (AIN#(0:149)_EF_READ_D@2)

Address: 7904

```
property ain2_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN2_NEGATIVE_CH (AIN#(0:13)_NEGATIVE_CH@2)

Address: 41002

```
property ain2_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN2_RANGE (AIN#(0:13)_RANGE@2)

Address: 40004

```
property ain2_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN2_RESOLUTION_INDEX (AIN#(0:13)_RESOLUTION_INDEX@2)

Address: 41502

```
property ain2_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN2_SETTLING_US (AIN#(0:13)_SETTLING_US@2)

Address: 42004

```
property ain3: float
```

Returns the voltage of the specified analog input.

Register: AIN3 (AIN#(0:13)@3)

Address: 6

```
property ain30_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@30)

Address: 9360

```
property ain30_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@30)

Address: 9660

```
property ain30_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@30)

Address: 9960

```
property ain30_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@30)

Address: 10260

```
property ain30_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@30)

Address: 10560

```
property ain30_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@30)

Address: 10860

```
property ain30_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@30)

Address: 11160

```
property ain30_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@30)

Address: 11460

```
property ain30_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@30)

Address: 11760

```
property ain30_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@30)

Address: 12060

```
property ain30_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN30_EF_INDEX (AIN#(0:149)_EF_INDEX@30)

Address: 9060

```
property ain30_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN30_EF_READ_A (AIN#(0:149)_EF_READ_A@30)

Address: 7060

```
property ain30_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN30_EF_READ_B (AIN#(0:149)_EF_READ_B@30)

Address: 7360

```
property ain30_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN30_EF_READ_C (AIN#(0:149)_EF_READ_C@30)

Address: 7660

```
property ain30_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN30_EF_READ_D (AIN#(0:149)_EF_READ_D@30)

Address: 7960

```
property ain31_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@31)

Address: 9362

```
property ain31_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@31)

Address: 9662

```
property ain31_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@31)

Address: 9962

```
property ain31_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@31)

Address: 10262

```
property ain31_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@31)

Address: 10562

```
property ain31_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@31)

Address: 10862

```
property ain31_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@31)

Address: 11162

```
property ain31_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@31)

Address: 11462

```
property ain31_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@31)

Address: 11762

```
property ain31_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@31)

Address: 12062

```
property ain31_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T;

25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN31_EF_INDEX (AIN#(0:149)_EF_INDEX@31)

Address: 9062

```
property ain31_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN31_EF_READ_A (AIN#(0:149)_EF_READ_A@31)

Address: 7062

```
property ain31_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN31_EF_READ_B (AIN#(0:149)_EF_READ_B@31)

Address: 7362

```
property ain31_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN31_EF_READ_C (AIN#(0:149)_EF_READ_C@31)

Address: 7662

```
property ain31_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN31_EF_READ_D (AIN#(0:149)_EF_READ_D@31)

Address: 7962

```
property ain32_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@32)

Address: 9364

```
property ain32_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@32)

Address: 9664

```
property ain32_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@32)

Address: 9964

```
property ain32_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@32)

Address: 10264

```
property ain32_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@32)

Address: 10564

```
property ain32_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@32)

Address: 10864

```
property ain32_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@32)

Address: 11164

```
property ain32_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@32)

Address: 11464

```
property ain32_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@32)

Address: 11764

```
property ain32_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@32)

Address: 12064

```
property ain32_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN32_EF_INDEX (AIN#(0:149)_EF_INDEX@32)

Address: 9064

```
property ain32_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN32_EF_READ_A (AIN#(0:149)_EF_READ_A@32)

Address: 7064

```
property ain32_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN32_EF_READ_B (AIN#(0:149)_EF_READ_B@32)

Address: 7364

```
property ain32_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN32_EF_READ_C (AIN#(0:149)_EF_READ_C@32)

Address: 7664

```
property ain32_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN32_EF_READ_D (AIN#(0:149)_EF_READ_D@32)

Address: 7964

```
property ain33_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@33)

Address: 9366

```
property ain33_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@33)

Address: 9666

```
property ain33_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@33)

Address: 9966

```
property ain33_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@33)

Address: 10266

```
property ain33_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@33)

Address: 10566

```
property ain33_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@33)

Address: 10866

```
property ain33_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@33)

Address: 11166

```
property ain33_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@33)

Address: 11466

```
property ain33_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@33)

Address: 11766

```
property ain33_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@33)

Address: 12066

```
property ain33_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN33_EF_INDEX (AIN#(0:149)_EF_INDEX@33)

Address: 9066

```
property ain33_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN33_EF_READ_A (AIN#(0:149)_EF_READ_A@33)

Address: 7066

```
property ain33_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN33_EF_READ_B (AIN#(0:149)_EF_READ_B@33)

Address: 7366

```
property ain33_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN33_EF_READ_C (AIN#(0:149)_EF_READ_C@33)

Address: 7666

```
property ain33_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN33_EF_READ_D (AIN#(0:149)_EF_READ_D@33)

Address: 7966

```
property ain34_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@34)

Address: 9368

```
property ain34_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@34)

Address: 9668

```
property ain34_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@34)

Address: 9968

```
property ain34_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@34)

Address: 10268

```
property ain34_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@34)

Address: 10568

```
property ain34_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@34)

Address: 10868

```
property ain34_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@34)

Address: 11168

```
property ain34_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@34)

Address: 11468

```
property ain34_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@34)

Address: 11768

```
property ain34_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@34)

Address: 12068

```
property ain34_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN34_EF_INDEX (AIN#(0:149)_EF_INDEX@34)

Address: 9068

```
property ain34_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN34_EF_READ_A (AIN#(0:149)_EF_READ_A@34)

Address: 7068

```
property ain34_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN34_EF_READ_B (AIN#(0:149)_EF_READ_B@34)

Address: 7368

```
property ain34_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN34_EF_READ_C (AIN#(0:149)_EF_READ_C@34)

Address: 7668

```
property ain34_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN34_EF_READ_D (AIN#(0:149)_EF_READ_D@34)

Address: 7968

```
property ain35_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@35)

Address: 9370

```
property ain35_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@35)

Address: 9670

```
property ain35_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@35)

Address: 9970

```
property ain35_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@35)

Address: 10270

```
property ain35_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@35)

Address: 10570

```
property ain35_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@35)

Address: 10870

```
property ain35_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@35)

Address: 11170

```
property ain35_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@35)

Address: 11470

```
property ain35_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@35)

Address: 11770

```
property ain35_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@35)

Address: 12070

```
property ain35_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN35_EF_INDEX (AIN#(0:149)_EF_INDEX@35)

Address: 9070

```
property ain35_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN35_EF_READ_A (AIN#(0:149)_EF_READ_A@35)

Address: 7070

```
property ain35_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN35_EF_READ_B (AIN#(0:149)_EF_READ_B@35)

Address: 7370

```
property ain35_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN35_EF_READ_C (AIN#(0:149)_EF_READ_C@35)

Address: 7670

```
property ain35_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN35_EF_READ_D (AIN#(0:149)_EF_READ_D@35)

Address: 7970

```
property ain36_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@36)

Address: 9372

```
property ain36_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@36)

Address: 9672

```
property ain36_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@36)

Address: 9972

```
property ain36_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@36)

Address: 10272

```
property ain36_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@36)

Address: 10572

```
property ain36_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@36)

Address: 10872

```
property ain36_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@36)

Address: 11172

```
property ain36_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@36)

Address: 11472

```
property ain36_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@36)

Address: 11772

```
property ain36_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@36)

Address: 12072

```
property ain36_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN36_EF_INDEX (AIN#(0:149)_EF_INDEX@36)

Address: 9072

```
property ain36_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN36_EF_READ_A (AIN#(0:149)_EF_READ_A@36)

Address: 7072

```
property ain36_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN36_EF_READ_B (AIN#(0:149)_EF_READ_B@36)

Address: 7372

```
property ain36_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN36_EF_READ_C (AIN#(0:149)_EF_READ_C@36)

Address: 7672

```
property ain36_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN36_EF_READ_D (AIN#(0:149)_EF_READ_D@36)

Address: 7972

```
property ain37_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@37)

Address: 9374

```
property ain37_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@37)

Address: 9674

```
property ain37_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@37)

Address: 9974

```
property ain37_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@37)

Address: 10274

```
property ain37_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@37)

Address: 10574

```
property ain37_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@37)

Address: 10874

```
property ain37_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@37)

Address: 11174

```
property ain37_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@37)

Address: 11474

```
property ain37_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@37)

Address: 11774

```
property ain37_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@37)

Address: 12074

```
property ain37_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN37_EF_INDEX (AIN#(0:149)_EF_INDEX@37)

Address: 9074

```
property ain37_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN37_EF_READ_A (AIN#(0:149)_EF_READ_A@37)

Address: 7074

```
property ain37_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN37_EF_READ_B (AIN#(0:149)_EF_READ_B@37)

Address: 7374

```
property ain37_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN37_EF_READ_C (AIN#(0:149)_EF_READ_C@37)

Address: 7674

```
property ain37_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN37_EF_READ_D (AIN#(0:149)_EF_READ_D@37)

Address: 7974

```
property ain38_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@38)

Address: 9376

```
property ain38_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@38)

Address: 9676

```
property ain38_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@38)

Address: 9976

```
property ain38_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@38)

Address: 10276

```
property ain38_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@38)

Address: 10576

```
property ain38_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@38)

Address: 10876

```
property ain38_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@38)

Address: 11176

```
property ain38_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@38)

Address: 11476

```
property ain38_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@38)

Address: 11776

```
property ain38_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@38)

Address: 12076

```
property ain38_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN38_EF_INDEX (AIN#(0:149)_EF_INDEX@38)

Address: 9076

```
property ain38_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN38_EF_READ_A (AIN#(0:149)_EF_READ_A@38)

Address: 7076

```
property ain38_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN38_EF_READ_B (AIN#(0:149)_EF_READ_B@38)

Address: 7376

```
property ain38_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN38_EF_READ_C (AIN#(0:149)_EF_READ_C@38)

Address: 7676

```
property ain38_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN38_EF_READ_D (AIN#(0:149)_EF_READ_D@38)

Address: 7976

```
property ain39_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@39)

Address: 9378

```
property ain39_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@39)

Address: 9678

```
property ain39_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@39)

Address: 9978

```
property ain39_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@39)

Address: 10278

```
property ain39_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@39)

Address: 10578

```
property ain39_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@39)

Address: 10878

```
property ain39_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@39)

Address: 11178

```
property ain39_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@39)

Address: 11478

```
property ain39_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@39)

Address: 11778

```
property ain39_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@39)

Address: 12078

```
property ain39_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN39_EF_INDEX (AIN#(0:149)_EF_INDEX@39)

Address: 9078

```
property ain39_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN39_EF_READ_A (AIN#(0:149)_EF_READ_A@39)

Address: 7078

```
property ain39_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN39_EF_READ_B (AIN#(0:149)_EF_READ_B@39)

Address: 7378

```
property ain39_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN39_EF_READ_C (AIN#(0:149)_EF_READ_C@39)

Address: 7678

```
property ain39_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN39_EF_READ_D (AIN#(0:149)_EF_READ_D@39)

Address: 7978

```
property ain3_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN3_BINARY (AIN#(0:13)_BINARY@3)

Address: 50006

```
property ain3_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@3)

Address: 9306

```
property ain3_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@3)

Address: 9606

```
property ain3_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@3)

Address: 9906

```
property ain3_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@3)

Address: 10206

```
property ain3_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@3)

Address: 10506

```
property ain3_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@3)

Address: 10806

```
property ain3_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@3)

Address: 11106

```
property ain3_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@3)

Address: 11406

```
property ain3_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@3)

Address: 11706

```
property ain3_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@3)

Address: 12006

```
property ain3_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN3_EF_INDEX (AIN#(0:149)_EF_INDEX@3)

Address: 9006

```
property ain3_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN3_EF_READ_A (AIN#(0:149)_EF_READ_A@3)

Address: 7006

```
property ain3_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN3_EF_READ_B (AIN#(0:149)_EF_READ_B@3)

Address: 7306

```
property ain3_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN3_EF_READ_C (AIN#(0:149)_EF_READ_C@3)

Address: 7606

```
property ain3_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN3_EF_READ_D (AIN#(0:149)_EF_READ_D@3)

Address: 7906

```
property ain3_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN3_NEGATIVE_CH (AIN#(0:13)_NEGATIVE_CH@3)

Address: 41003

```
property ain3_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN3_RANGE (AIN#(0:13)_RANGE@3)

Address: 40006

```
property ain3_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN3_RESOLUTION_INDEX (AIN#(0:13)_RESOLUTION_INDEX@3)

Address: 41503

```
property ain3_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN3_SETTLING_US (AIN#(0:13)_SETTLING_US@3)

Address: 42006

```
property ain4: float
```

Returns the voltage of the specified analog input.

Register: AIN4 (AIN#(0:13)@4)

Address: 8

```
property ain40_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@40)

Address: 9380

```
property ain40_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@40)

Address: 9680

```
property ain40_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@40)

Address: 9980

```
property ain40_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@40)

Address: 10280

```
property ain40_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@40)

Address: 10580

```
property ain40_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@40)

Address: 10880

```
property ain40_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@40)

Address: 11180

```
property ain40_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@40)

Address: 11480

```
property ain40_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@40)

Address: 11780

```
property ain40_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@40)

Address: 12080

```
property ain40_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN40_EF_INDEX (AIN#(0:149)_EF_INDEX@40)

Address: 9080

```
property ain40_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN40_EF_READ_A (AIN#(0:149)_EF_READ_A@40)

Address: 7080

```
property ain40_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN40_EF_READ_B (AIN#(0:149)_EF_READ_B@40)

Address: 7380

```
property ain40_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN40_EF_READ_C (AIN#(0:149)_EF_READ_C@40)

Address: 7680

```
property ain40_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN40_EF_READ_D (AIN#(0:149)_EF_READ_D@40)

Address: 7980

```
property ain41_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@41)

Address: 9382

```
property ain41_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@41)

Address: 9682

```
property ain41_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@41)

Address: 9982

```
property ain41_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@41)

Address: 10282

```
property ain41_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@41)

Address: 10582

```
property ain41_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@41)

Address: 10882

```
property ain41_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@41)

Address: 11182

```
property ain41_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@41)

Address: 11482

```
property ain41_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@41)

Address: 11782

```
property ain41_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@41)

Address: 12082

```
property ain41_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN41_EF_INDEX (AIN#(0:149)_EF_INDEX@41)

Address: 9082

```
property ain41_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN41_EF_READ_A (AIN#(0:149)_EF_READ_A@41)

Address: 7082

```
property ain41_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN41_EF_READ_B (AIN#(0:149)_EF_READ_B@41)

Address: 7382

```
property ain41_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN41_EF_READ_C (AIN#(0:149)_EF_READ_C@41)

Address: 7682

```
property ain41_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN41_EF_READ_D (AIN#(0:149)_EF_READ_D@41)

Address: 7982

```
property ain42_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@42)

Address: 9384

```
property ain42_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@42)

Address: 9684

```
property ain42_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@42)

Address: 9984

```
property ain42_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@42)

Address: 10284

```
property ain42_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@42)

Address: 10584

```
property ain42_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@42)

Address: 10884

```
property ain42_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@42)

Address: 11184

```
property ain42_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@42)

Address: 11484

```
property ain42_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@42)

Address: 11784

```
property ain42_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@42)

Address: 12084

```
property ain42_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J;

22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN42_EF_INDEX (AIN#(0:149)_EF_INDEX@42)

Address: 9084

```
property ain42_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN42_EF_READ_A (AIN#(0:149)_EF_READ_A@42)

Address: 7084

```
property ain42_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN42_EF_READ_B (AIN#(0:149)_EF_READ_B@42)

Address: 7384

```
property ain42_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN42_EF_READ_C (AIN#(0:149)_EF_READ_C@42)

Address: 7684

```
property ain42_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN42_EF_READ_D (AIN#(0:149)_EF_READ_D@42)

Address: 7984

```
property ain43_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@43)

Address: 9386

```
property ain43_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@43)

Address: 9686

```
property ain43_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@43)

Address: 9986

```
property ain43_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@43)

Address: 10286

```
property ain43_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@43)

Address: 10586

```
property ain43_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@43)

Address: 10886

```
property ain43_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@43)

Address: 11186

```
property ain43_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@43)

Address: 11486

```
property ain43_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@43)

Address: 11786

```
property ain43_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@43)

Address: 12086

```
property ain43_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN43_EF_INDEX (AIN#(0:149)_EF_INDEX@43)

Address: 9086

```
property ain43_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN43_EF_READ_A (AIN#(0:149)_EF_READ_A@43)

Address: 7086

```
property ain43_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN43_EF_READ_B (AIN#(0:149)_EF_READ_B@43)

Address: 7386

```
property ain43_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN43_EF_READ_C (AIN#(0:149)_EF_READ_C@43)

Address: 7686

```
property ain43_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN43_EF_READ_D (AIN#(0:149)_EF_READ_D@43)

Address: 7986

```
property ain44_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@44)

Address: 9388

```
property ain44_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@44)

Address: 9688

```
property ain44_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@44)

Address: 9988

```
property ain44_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@44)

Address: 10288

```
property ain44_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@44)

Address: 10588

```
property ain44_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@44)

Address: 10888

```
property ain44_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@44)

Address: 11188

```
property ain44_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@44)

Address: 11488

```
property ain44_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@44)

Address: 11788

```
property ain44_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@44)

Address: 12088

```
property ain44_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN44_EF_INDEX (AIN#(0:149)_EF_INDEX@44)

Address: 9088

```
property ain44_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN44_EF_READ_A (AIN#(0:149)_EF_READ_A@44)

Address: 7088

```
property ain44_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN44_EF_READ_B (AIN#(0:149)_EF_READ_B@44)

Address: 7388

```
property ain44_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN44_EF_READ_C (AIN#(0:149)_EF_READ_C@44)

Address: 7688

```
property ain44_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN44_EF_READ_D (AIN#(0:149)_EF_READ_D@44)

Address: 7988

```
property ain45_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@45)

Address: 9390

```
property ain45_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@45)

Address: 9690

```
property ain45_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@45)

Address: 9990

```
property ain45_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@45)

Address: 10290

```
property ain45_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@45)

Address: 10590

```
property ain45_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@45)

Address: 10890

```
property ain45_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@45)

Address: 11190

```
property ain45_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@45)

Address: 11490

```
property ain45_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@45)

Address: 11790

```
property ain45_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@45)

Address: 12090

```
property ain45_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN45_EF_INDEX (AIN#(0:149)_EF_INDEX@45)

Address: 9090

```
property ain45_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN45_EF_READ_A (AIN#(0:149)_EF_READ_A@45)

Address: 7090

```
property ain45_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN45_EF_READ_B (AIN#(0:149)_EF_READ_B@45)

Address: 7390

```
property ain45_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN45_EF_READ_C (AIN#(0:149)_EF_READ_C@45)

Address: 7690

```
property ain45_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN45_EF_READ_D (AIN#(0:149)_EF_READ_D@45)

Address: 7990

```
property ain46_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@46)

Address: 9392

```
property ain46_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@46)

Address: 9692

```
property ain46_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@46)

Address: 9992

```
property ain46_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@46)

Address: 10292

```
property ain46_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@46)

Address: 10592

```
property ain46_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@46)

Address: 10892

```
property ain46_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@46)

Address: 11192

```
property ain46_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@46)

Address: 11492

```
property ain46_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@46)

Address: 11792

```
property ain46_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@46)

Address: 12092

```
property ain46_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T;

25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN46_EF_INDEX (AIN#(0:149)_EF_INDEX@46)

Address: 9092

```
property ain46_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN46_EF_READ_A (AIN#(0:149)_EF_READ_A@46)

Address: 7092

```
property ain46_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN46_EF_READ_B (AIN#(0:149)_EF_READ_B@46)

Address: 7392

```
property ain46_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN46_EF_READ_C (AIN#(0:149)_EF_READ_C@46)

Address: 7692

```
property ain46_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN46_EF_READ_D (AIN#(0:149)_EF_READ_D@46)

Address: 7992

```
property ain47_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@47)

Address: 9394

```
property ain47_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@47)

Address: 9694

```
property ain47_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@47)

Address: 9994

```
property ain47_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@47)

Address: 10294

```
property ain47_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@47)

Address: 10594

```
property ain47_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@47)

Address: 10894

```
property ain47_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@47)

Address: 11194

```
property ain47_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@47)

Address: 11494

```
property ain47_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@47)

Address: 11794

```
property ain47_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@47)

Address: 12094

```
property ain47_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN47_EF_INDEX (AIN#(0:149)_EF_INDEX@47)

Address: 9094

```
property ain47_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN47_EF_READ_A (AIN#(0:149)_EF_READ_A@47)

Address: 7094

```
property ain47_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN47_EF_READ_B (AIN#(0:149)_EF_READ_B@47)

Address: 7394

```
property ain47_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN47_EF_READ_C (AIN#(0:149)_EF_READ_C@47)

Address: 7694

```
property ain47_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN47_EF_READ_D (AIN#(0:149)_EF_READ_D@47)

Address: 7994

```
property ain48: float
```

Returns the voltage of the specified analog input.

Register: AIN48 (AIN#(48:127)@48)

Address: 96

```
property ain48_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN48_BINARY (AIN#(48:127)_BINARY@48)

Address: 50096

```
property ain48_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@48)

Address: 9396

```
property ain48_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@48)

Address: 9696

```
property ain48_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@48)

Address: 9996

```
property ain48_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@48)

Address: 10296

```
property ain48_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@48)

Address: 10596

```
property ain48_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@48)

Address: 10896

```
property ain48_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@48)

Address: 11196

```
property ain48_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@48)

Address: 11496

```
property ain48_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@48)

Address: 11796

```
property ain48_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@48)

Address: 12096

```
property ain48_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN48_EF_INDEX (AIN#(0:149)_EF_INDEX@48)

Address: 9096

```
property ain48_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN48_EF_READ_A (AIN#(0:149)_EF_READ_A@48)

Address: 7096

```
property ain48_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN48_EF_READ_B (AIN#(0:149)_EF_READ_B@48)

Address: 7396

```
property ain48_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN48_EF_READ_C (AIN#(0:149)_EF_READ_C@48)

Address: 7696

```
property ain48_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN48_EF_READ_D (AIN#(0:149)_EF_READ_D@48)

Address: 7996

```
property ain48_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN48_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@48)

Address: 41048

```
property ain48_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN48_RANGE (AIN#(48:127)_RANGE@48)

Address: 40096

```
property ain48_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN48_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@48)

Address: 41548

```
property ain48_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN48_SETTLING_US (AIN#(48:127)_SETTLING_US@48)

Address: 42096

```
property ain49: float
```

Returns the voltage of the specified analog input.

Register: AIN49 (AIN#(48:127)@49)

Address: 98

```
property ain49_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN49_BINARY (AIN#(48:127)_BINARY@49)

Address: 50098

```
property ain49_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@49)

Address: 9398

```
property ain49_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@49)

Address: 9698

```
property ain49_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@49)

Address: 9998

```
property ain49_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@49)

Address: 10298

```
property ain49_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@49)

Address: 10598

```
property ain49_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@49)

Address: 10898

```
property ain49_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@49)

Address: 11198

```
property ain49_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@49)

Address: 11498

```
property ain49_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@49)

Address: 11798

```
property ain49_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@49)

Address: 12098

```
property ain49_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN49_EF_INDEX (AIN#(0:149)_EF_INDEX@49)

Address: 9098

```
property ain49_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN49_EF_READ_A (AIN#(0:149)_EF_READ_A@49)

Address: 7098

```
property ain49_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN49_EF_READ_B (AIN#(0:149)_EF_READ_B@49)

Address: 7398

```
property ain49_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN49_EF_READ_C (AIN#(0:149)_EF_READ_C@49)

Address: 7698

```
property ain49_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN49_EF_READ_D (AIN#(0:149)_EF_READ_D@49)

Address: 7998

```
property ain49_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN49_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@49)

Address: 41049

```
property ain49_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN49_RANGE (AIN#(48:127)_RANGE@49)

Address: 40098

```
property ain49_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN49_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@49)

Address: 41549

```
property ain49_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN49_SETTLING_US (AIN#(48:127)_SETTLING_US@49)

Address: 42098

```
property ain4_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN4_BINARY (AIN#(0:13)_BINARY@4)

Address: 50008

```
property ain4_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@4)

Address: 9308

```
property ain4_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@4)

Address: 9608

```
property ain4_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@4)

Address: 9908

```
property ain4_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@4)

Address: 10208

```
property ain4_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@4)

Address: 10508

```
property ain4_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@4)

Address: 10808

```
property ain4_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@4)

Address: 11108

```
property ain4_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@4)

Address: 11408

```
property ain4_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@4)

Address: 11708

```
property ain4_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@4)

Address: 12008

```
property ain4_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN4_EF_INDEX (AIN#(0:149)_EF_INDEX@4)

Address: 9008

```
property ain4_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN4_EF_READ_A (AIN#(0:149)_EF_READ_A@4)

Address: 7008

```
property ain4_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN4_EF_READ_B (AIN#(0:149)_EF_READ_B@4)

Address: 7308

```
property ain4_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN4_EF_READ_C (AIN#(0:149)_EF_READ_C@4)

Address: 7608

```
property ain4_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN4_EF_READ_D (AIN#(0:149)_EF_READ_D@4)

Address: 7908

```
property ain4_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN4_NEGATIVE_CH (AIN#(0:13)_NEGATIVE_CH@4)

Address: 41004

```
property ain4_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN4_RANGE (AIN#(0:13)_RANGE@4)

Address: 40008

```
property ain4_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN4_RESOLUTION_INDEX (AIN#(0:13)_RESOLUTION_INDEX@4)

Address: 41504

```
property ain4_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN4_SETTLING_US (AIN#(0:13)_SETTLING_US@4)

Address: 42008

```
property ain5: float
```

Returns the voltage of the specified analog input.

Register: AIN5 (AIN#(0:13)@5)

Address: 10

```
property ain50: float
```

Returns the voltage of the specified analog input.

Register: AIN50 (AIN#(48:127)@50)

Address: 100

```
property ain50_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN50_BINARY (AIN#(48:127)_BINARY@50)

Address: 50100

```
property ain50_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@50)

Address: 9400

```
property ain50_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@50)

Address: 9700

```
property ain50_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@50)

Address: 10000

```
property ain50_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@50)

Address: 10300

```
property ain50_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@50)

Address: 10600

```
property ain50_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@50)

Address: 10900

```
property ain50_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@50)

Address: 11200

```
property ain50_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@50)

Address: 11500

```
property ain50_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@50)

Address: 11800

```
property ain50_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@50)

Address: 12100

```
property ain50_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN50_EF_INDEX (AIN#(0:149)_EF_INDEX@50)

Address: 9100

```
property ain50_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN50_EF_READ_A (AIN#(0:149)_EF_READ_A@50)

Address: 7100

```
property ain50_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN50_EF_READ_B (AIN#(0:149)_EF_READ_B@50)

Address: 7400

```
property ain50_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN50_EF_READ_C (AIN#(0:149)_EF_READ_C@50)

Address: 7700

```
property ain50_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN50_EF_READ_D (AIN#(0:149)_EF_READ_D@50)

Address: 8000

```
property ain50_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN50_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@50)

Address: 41050

```
property ain50_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN50_RANGE (AIN#(48:127)_RANGE@50)

Address: 40100

```
property ain50_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN50_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@50)

Address: 41550

```
property ain50_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN50_SETTLING_US (AIN#(48:127)_SETTLING_US@50)

Address: 42100

```
property ain51: float
```

Returns the voltage of the specified analog input.

Register: AIN51 (AIN#(48:127)@51)

Address: 102

```
property ain51_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN51_BINARY (AIN#(48:127)_BINARY@51)

Address: 50102

```
property ain51_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@51)

Address: 9402

```
property ain51_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@51)

Address: 9702

```
property ain51_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@51)

Address: 10002

```
property ain51_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@51)

Address: 10302

```
property ain51_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@51)

Address: 10602

```
property ain51_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@51)

Address: 10902

```
property ain51_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@51)

Address: 11202

```
property ain51_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@51)

Address: 11502

```
property ain51_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@51)

Address: 11802

```
property ain51_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@51)

Address: 12102

```
property ain51_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN51_EF_INDEX (AIN#(0:149)_EF_INDEX@51)

Address: 9102

```
property ain51_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN51_EF_READ_A (AIN#(0:149)_EF_READ_A@51)

Address: 7102

```
property ain51_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN51_EF_READ_B (AIN#(0:149)_EF_READ_B@51)

Address: 7402

```
property ain51_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN51_EF_READ_C (AIN#(0:149)_EF_READ_C@51)

Address: 7702

```
property ain51_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN51_EF_READ_D (AIN#(0:149)_EF_READ_D@51)

Address: 8002

```
property ain51_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN51_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@51)

Address: 41051

```
property ain51_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN51_RANGE (AIN#(48:127)_RANGE@51)

Address: 40102

```
property ain51_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN51_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@51)

Address: 41551

```
property ain51_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN51_SETTLING_US (AIN#(48:127)_SETTLING_US@51)

Address: 42102

```
property ain52: float
```

Returns the voltage of the specified analog input.

Register: AIN52 (AIN#(48:127)@52)

Address: 104

```
property ain52_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN52_BINARY (AIN#(48:127)_BINARY@52)

Address: 50104

```
property ain52_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@52)

Address: 9404

```
property ain52_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@52)

Address: 9704

```
property ain52_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@52)

Address: 10004

```
property ain52_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@52)

Address: 10304

```
property ain52_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@52)

Address: 10604

```
property ain52_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@52)

Address: 10904

```
property ain52_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@52)

Address: 11204

```
property ain52_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@52)

Address: 11504

```
property ain52_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@52)

Address: 11804

```
property ain52_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@52)

Address: 12104

```
property ain52_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN52_EF_INDEX (AIN#(0:149)_EF_INDEX@52)

Address: 9104

```
property ain52_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN52_EF_READ_A (AIN#(0:149)_EF_READ_A@52)

Address: 7104

```
property ain52_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN52_EF_READ_B (AIN#(0:149)_EF_READ_B@52)

Address: 7404

```
property ain52_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN52_EF_READ_C (AIN#(0:149)_EF_READ_C@52)

Address: 7704

```
property ain52_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN52_EF_READ_D (AIN#(0:149)_EF_READ_D@52)

Address: 8004

```
property ain52_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN52_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@52)

Address: 41052

```
property ain52_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN52_RANGE (AIN#(48:127)_RANGE@52)

Address: 40104

```
property ain52_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN52_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@52)

Address: 41552

```
property ain52_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN52_SETTLING_US (AIN#(48:127)_SETTLING_US@52)

Address: 42104

```
property ain53: float
```

Returns the voltage of the specified analog input.

Register: AIN53 (AIN#(48:127)@53)

Address: 106

```
property ain53_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN53_BINARY (AIN#(48:127)_BINARY@53)

Address: 50106

```
property ain53_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@53)

Address: 9406

```
property ain53_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@53)

Address: 9706

```
property ain53_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@53)

Address: 10006

```
property ain53_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@53)

Address: 10306

```
property ain53_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@53)

Address: 10606

```
property ain53_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@53)

Address: 10906

```
property ain53_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@53)

Address: 11206

```
property ain53_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@53)

Address: 11506

```
property ain53_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@53)

Address: 11806

```
property ain53_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@53)

Address: 12106

```
property ain53_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN53_EF_INDEX (AIN#(0:149)_EF_INDEX@53)

Address: 9106

```
property ain53_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN53_EF_READ_A (AIN#(0:149)_EF_READ_A@53)

Address: 7106

```
property ain53_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN53_EF_READ_B (AIN#(0:149)_EF_READ_B@53)

Address: 7406

```
property ain53_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN53_EF_READ_C (AIN#(0:149)_EF_READ_C@53)

Address: 7706

```
property ain53_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN53_EF_READ_D (AIN#(0:149)_EF_READ_D@53)

Address: 8006

```
property ain53_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN53_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@53)

Address: 41053

```
property ain53_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN53_RANGE (AIN#(48:127)_RANGE@53)

Address: 40106

```
property ain53_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN53_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@53)

Address: 41553

```
property ain53_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN53_SETTLING_US (AIN#(48:127)_SETTLING_US@53)

Address: 42106

```
property ain54: float
```

Returns the voltage of the specified analog input.

Register: AIN54 (AIN#(48:127)@54)

Address: 108

```
property ain54_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN54_BINARY (AIN#(48:127)_BINARY@54)

Address: 50108

```
property ain54_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@54)

Address: 9408

```
property ain54_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@54)

Address: 9708

```
property ain54_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@54)

Address: 10008

```
property ain54_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@54)

Address: 10308

```
property ain54_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@54)

Address: 10608

```
property ain54_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@54)

Address: 10908

```
property ain54_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@54)

Address: 11208

```
property ain54_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@54)

Address: 11508

```
property ain54_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@54)

Address: 11808

```
property ain54_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@54)

Address: 12108

```
property ain54_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN54_EF_INDEX (AIN#(0:149)_EF_INDEX@54)

Address: 9108

```
property ain54_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN54_EF_READ_A (AIN#(0:149)_EF_READ_A@54)

Address: 7108

```
property ain54_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN54_EF_READ_B (AIN#(0:149)_EF_READ_B@54)

Address: 7408

```
property ain54_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN54_EF_READ_C (AIN#(0:149)_EF_READ_C@54)

Address: 7708

```
property ain54_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN54_EF_READ_D (AIN#(0:149)_EF_READ_D@54)

Address: 8008

```
property ain54_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN54_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@54)

Address: 41054

```
property ain54_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN54_RANGE (AIN#(48:127)_RANGE@54)

Address: 40108

```
property ain54_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN54_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@54)

Address: 41554

```
property ain54_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN54_SETTLING_US (AIN#(48:127)_SETTLING_US@54)

Address: 42108

```
property ain55: float
```

Returns the voltage of the specified analog input.

Register: AIN55 (AIN#(48:127)@55)

Address: 110

```
property ain55_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN55_BINARY (AIN#(48:127)_BINARY@55)

Address: 50110

```
property ain55_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@55)

Address: 9410

```
property ain55_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@55)

Address: 9710

```
property ain55_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@55)

Address: 10010

```
property ain55_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@55)

Address: 10310

```
property ain55_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@55)

Address: 10610

```
property ain55_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@55)

Address: 10910

```
property ain55_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@55)

Address: 11210

```
property ain55_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@55)

Address: 11510

```
property ain55_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@55)

Address: 11810

```
property ain55_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@55)

Address: 12110

```
property ain55_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN55_EF_INDEX (AIN#(0:149)_EF_INDEX@55)

Address: 9110

```
property ain55_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN55_EF_READ_A (AIN#(0:149)_EF_READ_A@55)

Address: 7110

```
property ain55_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN55_EF_READ_B (AIN#(0:149)_EF_READ_B@55)

Address: 7410

```
property ain55_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN55_EF_READ_C (AIN#(0:149)_EF_READ_C@55)

Address: 7710

```
property ain55_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN55_EF_READ_D (AIN#(0:149)_EF_READ_D@55)

Address: 8010

```
property ain55_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN55_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@55)

Address: 41055

```
property ain55_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN55_RANGE (AIN#(48:127)_RANGE@55)

Address: 40110

```
property ain55_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN55_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@55)

Address: 41555

```
property ain55_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN55_SETTLING_US (AIN#(48:127)_SETTLING_US@55)

Address: 42110

```
property ain56: float
```

Returns the voltage of the specified analog input.

Register: AIN56 (AIN#(48:127)@56)

Address: 112

```
property ain56_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN56_BINARY (AIN#(48:127)_BINARY@56)

Address: 50112

```
property ain56_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@56)

Address: 9412

```
property ain56_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@56)

Address: 9712

```
property ain56_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@56)

Address: 10012

```
property ain56_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@56)

Address: 10312

```
property ain56_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@56)

Address: 10612

```
property ain56_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@56)

Address: 10912

```
property ain56_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@56)

Address: 11212

```
property ain56_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@56)

Address: 11512

```
property ain56_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@56)

Address: 11812

```
property ain56_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@56)

Address: 12112

```
property ain56_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN56_EF_INDEX (AIN#(0:149)_EF_INDEX@56)

Address: 9112

```
property ain56_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN56_EF_READ_A (AIN#(0:149)_EF_READ_A@56)

Address: 7112

```
property ain56_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN56_EF_READ_B (AIN#(0:149)_EF_READ_B@56)

Address: 7412

```
property ain56_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN56_EF_READ_C (AIN#(0:149)_EF_READ_C@56)

Address: 7712

```
property ain56_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN56_EF_READ_D (AIN#(0:149)_EF_READ_D@56)

Address: 8012

```
property ain56_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN56_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@56)

Address: 41056

```
property ain56_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN56_RANGE (AIN#(48:127)_RANGE@56)

Address: 40112

```
property ain56_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN56_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@56)

Address: 41556

```
property ain56_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN56_SETTLING_US (AIN#(48:127)_SETTLING_US@56)

Address: 42112

```
property ain57: float
```

Returns the voltage of the specified analog input.

Register: AIN57 (AIN#(48:127)@57)

Address: 114

```
property ain57_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN57_BINARY (AIN#(48:127)_BINARY@57)

Address: 50114

```
property ain57_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@57)

Address: 9414

```
property ain57_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@57)

Address: 9714

```
property ain57_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@57)

Address: 10014

```
property ain57_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@57)

Address: 10314

```
property ain57_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@57)

Address: 10614

```
property ain57_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@57)

Address: 10914

```
property ain57_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@57)

Address: 11214

```
property ain57_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@57)

Address: 11514

```
property ain57_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@57)

Address: 11814

```
property ain57_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@57)

Address: 12114

```
property ain57_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN57_EF_INDEX (AIN#(0:149)_EF_INDEX@57)

Address: 9114

```
property ain57_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN57_EF_READ_A (AIN#(0:149)_EF_READ_A@57)

Address: 7114

```
property ain57_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN57_EF_READ_B (AIN#(0:149)_EF_READ_B@57)

Address: 7414

```
property ain57_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN57_EF_READ_C (AIN#(0:149)_EF_READ_C@57)

Address: 7714

```
property ain57_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN57_EF_READ_D (AIN#(0:149)_EF_READ_D@57)

Address: 8014

```
property ain57_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN57_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@57)

Address: 41057

```
property ain57_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN57_RANGE (AIN#(48:127)_RANGE@57)

Address: 40114

```
property ain57_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN57_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@57)

Address: 41557

```
property ain57_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN57_SETTLING_US (AIN#(48:127)_SETTLING_US@57)

Address: 42114

```
property ain58: float
```

Returns the voltage of the specified analog input.

Register: AIN58 (AIN#(48:127)@58)

Address: 116

```
property ain58_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN58_BINARY (AIN#(48:127)_BINARY@58)

Address: 50116

```
property ain58_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@58)

Address: 9416

```
property ain58_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@58)

Address: 9716

```
property ain58_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@58)

Address: 10016

```
property ain58_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@58)

Address: 10316

```
property ain58_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@58)

Address: 10616

```
property ain58_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@58)

Address: 10916

```
property ain58_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@58)

Address: 11216

```
property ain58_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@58)

Address: 11516

```
property ain58_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@58)

Address: 11816

```
property ain58_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@58)

Address: 12116

```
property ain58_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T;

25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN58_EF_INDEX (AIN#(0:149)_EF_INDEX@58)

Address: 9116

```
property ain58_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN58_EF_READ_A (AIN#(0:149)_EF_READ_A@58)

Address: 7116

```
property ain58_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN58_EF_READ_B (AIN#(0:149)_EF_READ_B@58)

Address: 7416

```
property ain58_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN58_EF_READ_C (AIN#(0:149)_EF_READ_C@58)

Address: 7716

```
property ain58_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN58_EF_READ_D (AIN#(0:149)_EF_READ_D@58)

Address: 8016

```
property ain58_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN58_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@58)

Address: 41058

```
property ain58_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN58_RANGE (AIN#(48:127)_RANGE@58)

Address: 40116

```
property ain58_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN58_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@58)

Address: 41558

```
property ain58_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN58_SETTLING_US (AIN#(48:127)_SETTLING_US@58)

Address: 42116

```
property ain59: float
```

Returns the voltage of the specified analog input.

Register: AIN59 (AIN#(48:127)@59)

Address: 118

```
property ain59_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN59_BINARY (AIN#(48:127)_BINARY@59)

Address: 50118

```
property ain59_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@59)

Address: 9418

```
property ain59_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@59)

Address: 9718

```
property ain59_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@59)

Address: 10018

```
property ain59_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@59)

Address: 10318

```
property ain59_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@59)

Address: 10618

```
property ain59_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@59)

Address: 10918

```
property ain59_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@59)

Address: 11218

```
property ain59_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@59)

Address: 11518

```
property ain59_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@59)

Address: 11818

```
property ain59_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@59)

Address: 12118

```
property ain59_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN59_EF_INDEX (AIN#(0:149)_EF_INDEX@59)

Address: 9118

```
property ain59_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN59_EF_READ_A (AIN#(0:149)_EF_READ_A@59)

Address: 7118

```
property ain59_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN59_EF_READ_B (AIN#(0:149)_EF_READ_B@59)

Address: 7418

```
property ain59_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN59_EF_READ_C (AIN#(0:149)_EF_READ_C@59)

Address: 7718

```
property ain59_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN59_EF_READ_D (AIN#(0:149)_EF_READ_D@59)

Address: 8018

```
property ain59_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN59_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@59)

Address: 41059

```
property ain59_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN59_RANGE (AIN#(48:127)_RANGE@59)

Address: 40118

```
property ain59_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN59_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@59)

Address: 41559

```
property ain59_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN59_SETTLING_US (AIN#(48:127)_SETTLING_US@59)

Address: 42118

```
property ain5_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN5_BINARY (AIN#(0:13)_BINARY@5)

Address: 50010

```
property ain5_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@5)

Address: 9310

```
property ain5_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@5)

Address: 9610

```
property ain5_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@5)

Address: 9910

```
property ain5_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@5)

Address: 10210

```
property ain5_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@5)

Address: 10510

```
property ain5_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@5)

Address: 10810

```
property ain5_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@5)

Address: 11110

```
property ain5_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@5)

Address: 11410

```
property ain5_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@5)

Address: 11710

```
property ain5_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@5)

Address: 12010

```
property ain5_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN5_EF_INDEX (AIN#(0:149)_EF_INDEX@5)

Address: 9010

```
property ain5_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN5_EF_READ_A (AIN#(0:149)_EF_READ_A@5)

Address: 7010

```
property ain5_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN5_EF_READ_B (AIN#(0:149)_EF_READ_B@5)

Address: 7310

```
property ain5_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN5_EF_READ_C (AIN#(0:149)_EF_READ_C@5)

Address: 7610

```
property ain5_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN5_EF_READ_D (AIN#(0:149)_EF_READ_D@5)

Address: 7910

```
property ain5_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN5_NEGATIVE_CH (AIN#(0:13)_NEGATIVE_CH@5)

Address: 41005

```
property ain5_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN5_RANGE (AIN#(0:13)_RANGE@5)

Address: 40010

```
property ain5_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN5_RESOLUTION_INDEX (AIN#(0:13)_RESOLUTION_INDEX@5)

Address: 41505

```
property ain5_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN5_SETTLING_US (AIN#(0:13)_SETTLING_US@5)

Address: 42010

```
property ain6: float
```

Returns the voltage of the specified analog input.

Register: AIN6 (AIN#(0:13)@6)

Address: 12

```
property ain60: float
```

Returns the voltage of the specified analog input.

Register: AIN60 (AIN#(48:127)@60)

Address: 120

```
property ain60_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN60_BINARY (AIN#(48:127)_BINARY@60)

Address: 50120

```
property ain60_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@60)

Address: 9420

```
property ain60_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@60)

Address: 9720

```
property ain60_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@60)

Address: 10020

```
property ain60_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@60)

Address: 10320

```
property ain60_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@60)

Address: 10620

```
property ain60_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@60)

Address: 10920

```
property ain60_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@60)

Address: 11220

```
property ain60_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@60)

Address: 11520

```
property ain60_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@60)

Address: 11820

```
property ain60_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@60)

Address: 12120

```
property ain60_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN60_EF_INDEX (AIN#(0:149)_EF_INDEX@60)

Address: 9120

```
property ain60_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN60_EF_READ_A (AIN#(0:149)_EF_READ_A@60)

Address: 7120

```
property ain60_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN60_EF_READ_B (AIN#(0:149)_EF_READ_B@60)

Address: 7420

```
property ain60_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN60_EF_READ_C (AIN#(0:149)_EF_READ_C@60)

Address: 7720

```
property ain60_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN60_EF_READ_D (AIN#(0:149)_EF_READ_D@60)

Address: 8020

```
property ain60_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN60_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@60)

Address: 41060

```
property ain60_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN60_RANGE (AIN#(48:127)_RANGE@60)

Address: 40120

```
property ain60_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN60_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@60)

Address: 41560

```
property ain60_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN60_SETTLING_US (AIN#(48:127)_SETTLING_US@60)

Address: 42120

```
property ain61: float
```

Returns the voltage of the specified analog input.

Register: AIN61 (AIN#(48:127)@61)

Address: 122

```
property ain61_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN61_BINARY (AIN#(48:127)_BINARY@61)

Address: 50122

```
property ain61_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@61)

Address: 9422

```
property ain61_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@61)

Address: 9722

```
property ain61_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@61)

Address: 10022

```
property ain61_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@61)

Address: 10322

```
property ain61_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@61)

Address: 10622

```
property ain61_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@61)

Address: 10922

```
property ain61_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@61)

Address: 11222

```
property ain61_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@61)

Address: 11522

```
property ain61_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@61)

Address: 11822

```
property ain61_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@61)

Address: 12122

```
property ain61_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN61_EF_INDEX (AIN#(0:149)_EF_INDEX@61)

Address: 9122

```
property ain61_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN61_EF_READ_A (AIN#(0:149)_EF_READ_A@61)

Address: 7122

```
property ain61_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN61_EF_READ_B (AIN#(0:149)_EF_READ_B@61)

Address: 7422

```
property ain61_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN61_EF_READ_C (AIN#(0:149)_EF_READ_C@61)

Address: 7722

```
property ain61_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN61_EF_READ_D (AIN#(0:149)_EF_READ_D@61)

Address: 8022

```
property ain61_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN61_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@61)

Address: 41061

```
property ain61_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN61_RANGE (AIN#(48:127)_RANGE@61)

Address: 40122

```
property ain61_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN61_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@61)

Address: 41561

```
property ain61_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN61_SETTLING_US (AIN#(48:127)_SETTLING_US@61)

Address: 42122

```
property ain62: float
```

Returns the voltage of the specified analog input.

Register: AIN62 (AIN#(48:127)@62)

Address: 124

```
property ain62_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN62_BINARY (AIN#(48:127)_BINARY@62)

Address: 50124

```
property ain62_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@62)

Address: 9424

```
property ain62_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@62)

Address: 9724

```
property ain62_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@62)

Address: 10024

```
property ain62_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@62)

Address: 10324

```
property ain62_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@62)

Address: 10624

```
property ain62_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@62)

Address: 10924

```
property ain62_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@62)

Address: 11224

```
property ain62_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@62)

Address: 11524

```
property ain62_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@62)

Address: 11824

```
property ain62_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@62)

Address: 12124

```
property ain62_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN62_EF_INDEX (AIN#(0:149)_EF_INDEX@62)

Address: 9124

```
property ain62_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN62_EF_READ_A (AIN#(0:149)_EF_READ_A@62)

Address: 7124

```
property ain62_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN62_EF_READ_B (AIN#(0:149)_EF_READ_B@62)

Address: 7424

```
property ain62_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN62_EF_READ_C (AIN#(0:149)_EF_READ_C@62)

Address: 7724

```
property ain62_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN62_EF_READ_D (AIN#(0:149)_EF_READ_D@62)

Address: 8024

```
property ain62_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN62_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@62)

Address: 41062

```
property ain62_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN62_RANGE (AIN#(48:127)_RANGE@62)

Address: 40124

```
property ain62_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN62_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@62)

Address: 41562

```
property ain62_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN62_SETTLING_US (AIN#(48:127)_SETTLING_US@62)

Address: 42124

```
property ain63: float
```

Returns the voltage of the specified analog input.

Register: AIN63 (AIN#(48:127)@63)

Address: 126

```
property ain63_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN63_BINARY (AIN#(48:127)_BINARY@63)

Address: 50126

```
property ain63_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@63)

Address: 9426

```
property ain63_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@63)

Address: 9726

```
property ain63_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@63)

Address: 10026

```
property ain63_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@63)

Address: 10326

```
property ain63_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@63)

Address: 10626

```
property ain63_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@63)

Address: 10926

```
property ain63_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@63)

Address: 11226

```
property ain63_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@63)

Address: 11526

```
property ain63_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@63)

Address: 11826

```
property ain63_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@63)

Address: 12126

```
property ain63_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN63_EF_INDEX (AIN#(0:149)_EF_INDEX@63)

Address: 9126

```
property ain63_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN63_EF_READ_A (AIN#(0:149)_EF_READ_A@63)

Address: 7126

```
property ain63_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN63_EF_READ_B (AIN#(0:149)_EF_READ_B@63)

Address: 7426

```
property ain63_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN63_EF_READ_C (AIN#(0:149)_EF_READ_C@63)

Address: 7726

```
property ain63_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN63_EF_READ_D (AIN#(0:149)_EF_READ_D@63)

Address: 8026

```
property ain63_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN63_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@63)

Address: 41063

```
property ain63_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN63_RANGE (AIN#(48:127)_RANGE@63)

Address: 40126

```
property ain63_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN63_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@63)

Address: 41563

```
property ain63_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN63_SETTLING_US (AIN#(48:127)_SETTLING_US@63)

Address: 42126

```
property ain64: float
```

Returns the voltage of the specified analog input.

Register: AIN64 (AIN#(48:127)@64)

Address: 128

```
property ain64_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN64_BINARY (AIN#(48:127)_BINARY@64)

Address: 50128

```
property ain64_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@64)

Address: 9428

```
property ain64_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@64)

Address: 9728

```
property ain64_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@64)

Address: 10028

```
property ain64_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@64)

Address: 10328

```
property ain64_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@64)

Address: 10628

```
property ain64_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@64)

Address: 10928

```
property ain64_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@64)

Address: 11228

```
property ain64_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@64)

Address: 11528

```
property ain64_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@64)

Address: 11828

```
property ain64_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@64)

Address: 12128

```
property ain64_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN64_EF_INDEX (AIN#(0:149)_EF_INDEX@64)

Address: 9128

```
property ain64_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN64_EF_READ_A (AIN#(0:149)_EF_READ_A@64)

Address: 7128

```
property ain64_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN64_EF_READ_B (AIN#(0:149)_EF_READ_B@64)

Address: 7428

```
property ain64_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN64_EF_READ_C (AIN#(0:149)_EF_READ_C@64)

Address: 7728

```
property ain64_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN64_EF_READ_D (AIN#(0:149)_EF_READ_D@64)

Address: 8028

```
property ain64_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN64_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@64)

Address: 41064

```
property ain64_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN64_RANGE (AIN#(48:127)_RANGE@64)

Address: 40128

```
property ain64_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN64_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@64)

Address: 41564

```
property ain64_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN64_SETTLING_US (AIN#(48:127)_SETTLING_US@64)

Address: 42128

```
property ain65: float
```

Returns the voltage of the specified analog input.

Register: AIN65 (AIN#(48:127)@65)

Address: 130

```
property ain65_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN65_BINARY (AIN#(48:127)_BINARY@65)

Address: 50130

```
property ain65_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@65)

Address: 9430

```
property ain65_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@65)

Address: 9730

```
property ain65_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@65)

Address: 10030

```
property ain65_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@65)

Address: 10330

```
property ain65_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@65)

Address: 10630

```
property ain65_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@65)

Address: 10930

```
property ain65_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@65)

Address: 11230

```
property ain65_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@65)

Address: 11530

```
property ain65_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@65)

Address: 11830

```
property ain65_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@65)

Address: 12130

```
property ain65_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN65_EF_INDEX (AIN#(0:149)_EF_INDEX@65)

Address: 9130

```
property ain65_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN65_EF_READ_A (AIN#(0:149)_EF_READ_A@65)

Address: 7130

```
property ain65_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN65_EF_READ_B (AIN#(0:149)_EF_READ_B@65)

Address: 7430

```
property ain65_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN65_EF_READ_C (AIN#(0:149)_EF_READ_C@65)

Address: 7730

```
property ain65_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN65_EF_READ_D (AIN#(0:149)_EF_READ_D@65)

Address: 8030

```
property ain65_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN65_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@65)

Address: 41065

```
property ain65_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN65_RANGE (AIN#(48:127)_RANGE@65)

Address: 40130

```
property ain65_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN65_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@65)

Address: 41565

```
property ain65_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN65_SETTLING_US (AIN#(48:127)_SETTLING_US@65)

Address: 42130

```
property ain66: float
```

Returns the voltage of the specified analog input.

Register: AIN66 (AIN#(48:127)@66)

Address: 132

```
property ain66_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN66_BINARY (AIN#(48:127)_BINARY@66)

Address: 50132

```
property ain66_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@66)

Address: 9432

```
property ain66_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@66)

Address: 9732

```
property ain66_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@66)

Address: 10032

```
property ain66_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@66)

Address: 10332

```
property ain66_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@66)

Address: 10632

```
property ain66_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@66)

Address: 10932

```
property ain66_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@66)

Address: 11232

```
property ain66_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@66)

Address: 11532

```
property ain66_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@66)

Address: 11832

```
property ain66_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@66)

Address: 12132

```
property ain66_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN66_EF_INDEX (AIN#(0:149)_EF_INDEX@66)

Address: 9132

```
property ain66_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN66_EF_READ_A (AIN#(0:149)_EF_READ_A@66)

Address: 7132

```
property ain66_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN66_EF_READ_B (AIN#(0:149)_EF_READ_B@66)

Address: 7432

```
property ain66_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN66_EF_READ_C (AIN#(0:149)_EF_READ_C@66)

Address: 7732

```
property ain66_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN66_EF_READ_D (AIN#(0:149)_EF_READ_D@66)

Address: 8032

```
property ain66_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN66_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@66)

Address: 41066

```
property ain66_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN66_RANGE (AIN#(48:127)_RANGE@66)

Address: 40132

```
property ain66_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN66_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@66)

Address: 41566

```
property ain66_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN66_SETTLING_US (AIN#(48:127)_SETTLING_US@66)

Address: 42132

```
property ain67: float
```

Returns the voltage of the specified analog input.

Register: AIN67 (AIN#(48:127)@67)

Address: 134

```
property ain67_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN67_BINARY (AIN#(48:127)_BINARY@67)

Address: 50134

```
property ain67_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@67)

Address: 9434

```
property ain67_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@67)

Address: 9734

```
property ain67_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@67)

Address: 10034

```
property ain67_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@67)

Address: 10334

```
property ain67_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@67)

Address: 10634

```
property ain67_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@67)

Address: 10934

```
property ain67_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@67)

Address: 11234

```
property ain67_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@67)

Address: 11534

```
property ain67_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@67)

Address: 11834

```
property ain67_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@67)

Address: 12134

```
property ain67_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN67_EF_INDEX (AIN#(0:149)_EF_INDEX@67)

Address: 9134

```
property ain67_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN67_EF_READ_A (AIN#(0:149)_EF_READ_A@67)

Address: 7134

```
property ain67_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN67_EF_READ_B (AIN#(0:149)_EF_READ_B@67)

Address: 7434

```
property ain67_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN67_EF_READ_C (AIN#(0:149)_EF_READ_C@67)

Address: 7734

```
property ain67_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN67_EF_READ_D (AIN#(0:149)_EF_READ_D@67)

Address: 8034

```
property ain67_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN67_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@67)

Address: 41067

```
property ain67_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN67_RANGE (AIN#(48:127)_RANGE@67)

Address: 40134

```
property ain67_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN67_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@67)

Address: 41567

```
property ain67_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN67_SETTLING_US (AIN#(48:127)_SETTLING_US@67)

Address: 42134

```
property ain68: float
```

Returns the voltage of the specified analog input.

Register: AIN68 (AIN#(48:127)@68)

Address: 136

```
property ain68_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN68_BINARY (AIN#(48:127)_BINARY@68)

Address: 50136

```
property ain68_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@68)

Address: 9436

```
property ain68_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@68)

Address: 9736

```
property ain68_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@68)

Address: 10036

```
property ain68_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@68)

Address: 10336

```
property ain68_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@68)

Address: 10636

```
property ain68_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@68)

Address: 10936

```
property ain68_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@68)

Address: 11236

```
property ain68_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@68)

Address: 11536

```
property ain68_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@68)

Address: 11836

```
property ain68_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@68)

Address: 12136

```
property ain68_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T;

25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN68_EF_INDEX (AIN#(0:149)_EF_INDEX@68)

Address: 9136

```
property ain68_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN68_EF_READ_A (AIN#(0:149)_EF_READ_A@68)

Address: 7136

```
property ain68_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN68_EF_READ_B (AIN#(0:149)_EF_READ_B@68)

Address: 7436

```
property ain68_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN68_EF_READ_C (AIN#(0:149)_EF_READ_C@68)

Address: 7736

```
property ain68_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN68_EF_READ_D (AIN#(0:149)_EF_READ_D@68)

Address: 8036

```
property ain68_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN68_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@68)

Address: 41068

```
property ain68_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN68_RANGE (AIN#(48:127)_RANGE@68)

Address: 40136

```
property ain68_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN68_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@68)

Address: 41568

```
property ain68_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN68_SETTLING_US (AIN#(48:127)_SETTLING_US@68)

Address: 42136

```
property ain69: float
```

Returns the voltage of the specified analog input.

Register: AIN69 (AIN#(48:127)@69)

Address: 138

```
property ain69_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN69_BINARY (AIN#(48:127)_BINARY@69)

Address: 50138

```
property ain69_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@69)

Address: 9438

```
property ain69_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@69)

Address: 9738

```
property ain69_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@69)

Address: 10038

```
property ain69_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@69)

Address: 10338

```
property ain69_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@69)

Address: 10638

```
property ain69_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@69)

Address: 10938

```
property ain69_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@69)

Address: 11238

```
property ain69_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@69)

Address: 11538

```
property ain69_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@69)

Address: 11838

```
property ain69_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@69)

Address: 12138

```
property ain69_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN69_EF_INDEX (AIN#(0:149)_EF_INDEX@69)

Address: 9138

```
property ain69_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN69_EF_READ_A (AIN#(0:149)_EF_READ_A@69)

Address: 7138

```
property ain69_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN69_EF_READ_B (AIN#(0:149)_EF_READ_B@69)

Address: 7438

```
property ain69_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN69_EF_READ_C (AIN#(0:149)_EF_READ_C@69)

Address: 7738

```
property ain69_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN69_EF_READ_D (AIN#(0:149)_EF_READ_D@69)

Address: 8038

```
property ain69_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN69_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@69)

Address: 41069

```
property ain69_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN69_RANGE (AIN#(48:127)_RANGE@69)

Address: 40138

```
property ain69_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN69_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@69)

Address: 41569

```
property ain69_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN69_SETTLING_US (AIN#(48:127)_SETTLING_US@69)

Address: 42138

```
property ain6_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN6_BINARY (AIN#(0:13)_BINARY@6)

Address: 50012

```
property ain6_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@6)

Address: 9312

```
property ain6_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@6)

Address: 9612

```
property ain6_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@6)

Address: 9912

```
property ain6_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@6)

Address: 10212

```
property ain6_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@6)

Address: 10512

```
property ain6_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@6)

Address: 10812

```
property ain6_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@6)

Address: 11112

```
property ain6_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@6)

Address: 11412

```
property ain6_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@6)

Address: 11712

```
property ain6_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@6)

Address: 12012

```
property ain6_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN6_EF_INDEX (AIN#(0:149)_EF_INDEX@6)

Address: 9012

```
property ain6_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN6_EF_READ_A (AIN#(0:149)_EF_READ_A@6)

Address: 7012

```
property ain6_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN6_EF_READ_B (AIN#(0:149)_EF_READ_B@6)

Address: 7312

```
property ain6_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN6_EF_READ_C (AIN#(0:149)_EF_READ_C@6)

Address: 7612

```
property ain6_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN6_EF_READ_D (AIN#(0:149)_EF_READ_D@6)

Address: 7912

```
property ain6_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN6_NEGATIVE_CH (AIN#(0:13)_NEGATIVE_CH@6)

Address: 41006

```
property ain6_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN6_RANGE (AIN#(0:13)_RANGE@6)

Address: 40012

```
property ain6_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN6_RESOLUTION_INDEX (AIN#(0:13)_RESOLUTION_INDEX@6)

Address: 41506

```
property ain6_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN6_SETTLING_US (AIN#(0:13)_SETTLING_US@6)

Address: 42012

```
property ain7: float
```

Returns the voltage of the specified analog input.

Register: AIN7 (AIN#(0:13)@7)

Address: 14

```
property ain70: float
```

Returns the voltage of the specified analog input.

Register: AIN70 (AIN#(48:127)@70)

Address: 140

```
property ain70_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN70_BINARY (AIN#(48:127)_BINARY@70)

Address: 50140

```
property ain70_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@70)

Address: 9440

```
property ain70_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@70)

Address: 9740

```
property ain70_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@70)

Address: 10040

```
property ain70_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@70)

Address: 10340

```
property ain70_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@70)

Address: 10640

```
property ain70_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@70)

Address: 10940

```
property ain70_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@70)

Address: 11240

```
property ain70_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@70)

Address: 11540

```
property ain70_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@70)

Address: 11840

```
property ain70_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@70)

Address: 12140

```
property ain70_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN70_EF_INDEX (AIN#(0:149)_EF_INDEX@70)

Address: 9140

```
property ain70_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN70_EF_READ_A (AIN#(0:149)_EF_READ_A@70)

Address: 7140

```
property ain70_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN70_EF_READ_B (AIN#(0:149)_EF_READ_B@70)

Address: 7440

```
property ain70_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN70_EF_READ_C (AIN#(0:149)_EF_READ_C@70)

Address: 7740

```
property ain70_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN70_EF_READ_D (AIN#(0:149)_EF_READ_D@70)

Address: 8040

```
property ain70_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN70_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@70)

Address: 41070

```
property ain70_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN70_RANGE (AIN#(48:127)_RANGE@70)

Address: 40140

```
property ain70_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN70_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@70)

Address: 41570

```
property ain70_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN70_SETTLING_US (AIN#(48:127)_SETTLING_US@70)

Address: 42140

```
property ain71: float
```

Returns the voltage of the specified analog input.

Register: AIN71 (AIN#(48:127)@71)

Address: 142

```
property ain71_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN71_BINARY (AIN#(48:127)_BINARY@71)

Address: 50142

```
property ain71_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@71)

Address: 9442

```
property ain71_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@71)

Address: 9742

```
property ain71_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@71)

Address: 10042

```
property ain71_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@71)

Address: 10342

```
property ain71_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@71)

Address: 10642

```
property ain71_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@71)

Address: 10942

```
property ain71_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@71)

Address: 11242

```
property ain71_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@71)

Address: 11542

```
property ain71_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@71)

Address: 11842

```
property ain71_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@71)

Address: 12142

```
property ain71_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN71_EF_INDEX (AIN#(0:149)_EF_INDEX@71)

Address: 9142

```
property ain71_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN71_EF_READ_A (AIN#(0:149)_EF_READ_A@71)

Address: 7142

```
property ain71_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN71_EF_READ_B (AIN#(0:149)_EF_READ_B@71)

Address: 7442

```
property ain71_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN71_EF_READ_C (AIN#(0:149)_EF_READ_C@71)

Address: 7742

```
property ain71_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN71_EF_READ_D (AIN#(0:149)_EF_READ_D@71)

Address: 8042

```
property ain71_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN71_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@71)

Address: 41071

```
property ain71_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN71_RANGE (AIN#(48:127)_RANGE@71)

Address: 40142

```
property ain71_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN71_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@71)

Address: 41571

```
property ain71_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN71_SETTLING_US (AIN#(48:127)_SETTLING_US@71)

Address: 42142

```
property ain72: float
```

Returns the voltage of the specified analog input.

Register: AIN72 (AIN#(48:127)@72)

Address: 144

```
property ain72_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN72_BINARY (AIN#(48:127)_BINARY@72)

Address: 50144

```
property ain72_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@72)

Address: 9444

```
property ain72_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@72)

Address: 9744

```
property ain72_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@72)

Address: 10044

```
property ain72_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@72)

Address: 10344

```
property ain72_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@72)

Address: 10644

```
property ain72_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@72)

Address: 10944

```
property ain72_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@72)

Address: 11244

```
property ain72_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@72)

Address: 11544

```
property ain72_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@72)

Address: 11844

```
property ain72_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@72)

Address: 12144

```
property ain72_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN72_EF_INDEX (AIN#(0:149)_EF_INDEX@72)

Address: 9144

```
property ain72_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN72_EF_READ_A (AIN#(0:149)_EF_READ_A@72)

Address: 7144

```
property ain72_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN72_EF_READ_B (AIN#(0:149)_EF_READ_B@72)

Address: 7444

```
property ain72_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN72_EF_READ_C (AIN#(0:149)_EF_READ_C@72)

Address: 7744

```
property ain72_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN72_EF_READ_D (AIN#(0:149)_EF_READ_D@72)

Address: 8044

```
property ain72_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN72_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@72)

Address: 41072

```
property ain72_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN72_RANGE (AIN#(48:127)_RANGE@72)

Address: 40144

```
property ain72_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN72_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@72)

Address: 41572

```
property ain72_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN72_SETTLING_US (AIN#(48:127)_SETTLING_US@72)

Address: 42144

```
property ain73: float
```

Returns the voltage of the specified analog input.

Register: AIN73 (AIN#(48:127)@73)

Address: 146

```
property ain73_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN73_BINARY (AIN#(48:127)_BINARY@73)

Address: 50146

```
property ain73_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@73)

Address: 9446

```
property ain73_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@73)

Address: 9746

```
property ain73_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@73)

Address: 10046

```
property ain73_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@73)

Address: 10346

```
property ain73_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@73)

Address: 10646

```
property ain73_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@73)

Address: 10946

```
property ain73_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@73)

Address: 11246

```
property ain73_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@73)

Address: 11546

```
property ain73_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@73)

Address: 11846

```
property ain73_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@73)

Address: 12146

```
property ain73_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN73_EF_INDEX (AIN#(0:149)_EF_INDEX@73)

Address: 9146

```
property ain73_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN73_EF_READ_A (AIN#(0:149)_EF_READ_A@73)

Address: 7146

```
property ain73_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN73_EF_READ_B (AIN#(0:149)_EF_READ_B@73)

Address: 7446

```
property ain73_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN73_EF_READ_C (AIN#(0:149)_EF_READ_C@73)

Address: 7746

```
property ain73_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN73_EF_READ_D (AIN#(0:149)_EF_READ_D@73)

Address: 8046

```
property ain73_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN73_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@73)

Address: 41073

```
property ain73_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN73_RANGE (AIN#(48:127)_RANGE@73)

Address: 40146

```
property ain73_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN73_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@73)

Address: 41573

```
property ain73_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN73_SETTLING_US (AIN#(48:127)_SETTLING_US@73)

Address: 42146

```
property ain74: float
```

Returns the voltage of the specified analog input.

Register: AIN74 (AIN#(48:127)@74)

Address: 148

```
property ain74_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN74_BINARY (AIN#(48:127)_BINARY@74)

Address: 50148

```
property ain74_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@74)

Address: 9448

```
property ain74_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@74)

Address: 9748

```
property ain74_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@74)

Address: 10048

```
property ain74_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@74)

Address: 10348

```
property ain74_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@74)

Address: 10648

```
property ain74_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@74)

Address: 10948

```
property ain74_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@74)

Address: 11248

```
property ain74_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@74)

Address: 11548

```
property ain74_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@74)

Address: 11848

```
property ain74_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@74)

Address: 12148

```
property ain74_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN74_EF_INDEX (AIN#(0:149)_EF_INDEX@74)

Address: 9148

```
property ain74_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN74_EF_READ_A (AIN#(0:149)_EF_READ_A@74)

Address: 7148

```
property ain74_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN74_EF_READ_B (AIN#(0:149)_EF_READ_B@74)

Address: 7448

```
property ain74_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN74_EF_READ_C (AIN#(0:149)_EF_READ_C@74)

Address: 7748

```
property ain74_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN74_EF_READ_D (AIN#(0:149)_EF_READ_D@74)

Address: 8048

```
property ain74_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN74_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@74)

Address: 41074

```
property ain74_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN74_RANGE (AIN#(48:127)_RANGE@74)

Address: 40148

```
property ain74_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN74_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@74)

Address: 41574

```
property ain74_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN74_SETTLING_US (AIN#(48:127)_SETTLING_US@74)

Address: 42148

```
property ain75: float
```

Returns the voltage of the specified analog input.

Register: AIN75 (AIN#(48:127)@75)

Address: 150

```
property ain75_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN75_BINARY (AIN#(48:127)_BINARY@75)

Address: 50150

```
property ain75_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@75)

Address: 9450

```
property ain75_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@75)

Address: 9750

```
property ain75_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@75)

Address: 10050

```
property ain75_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@75)

Address: 10350

```
property ain75_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@75)

Address: 10650

```
property ain75_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@75)

Address: 10950

```
property ain75_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@75)

Address: 11250

```
property ain75_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@75)

Address: 11550

```
property ain75_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@75)

Address: 11850

```
property ain75_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@75)

Address: 12150

```
property ain75_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN75_EF_INDEX (AIN#(0:149)_EF_INDEX@75)

Address: 9150

```
property ain75_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN75_EF_READ_A (AIN#(0:149)_EF_READ_A@75)

Address: 7150

```
property ain75_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN75_EF_READ_B (AIN#(0:149)_EF_READ_B@75)

Address: 7450

```
property ain75_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN75_EF_READ_C (AIN#(0:149)_EF_READ_C@75)

Address: 7750

```
property ain75_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN75_EF_READ_D (AIN#(0:149)_EF_READ_D@75)

Address: 8050

```
property ain75_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN75_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@75)

Address: 41075

```
property ain75_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN75_RANGE (AIN#(48:127)_RANGE@75)

Address: 40150

```
property ain75_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN75_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@75)

Address: 41575

```
property ain75_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN75_SETTLING_US (AIN#(48:127)_SETTLING_US@75)

Address: 42150

```
property ain76: float
```

Returns the voltage of the specified analog input.

Register: AIN76 (AIN#(48:127)@76)

Address: 152

```
property ain76_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN76_BINARY (AIN#(48:127)_BINARY@76)

Address: 50152

```
property ain76_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@76)

Address: 9452

```
property ain76_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@76)

Address: 9752

```
property ain76_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@76)

Address: 10052

```
property ain76_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@76)

Address: 10352

```
property ain76_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@76)

Address: 10652

```
property ain76_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@76)

Address: 10952

```
property ain76_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@76)

Address: 11252

```
property ain76_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@76)

Address: 11552

```
property ain76_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@76)

Address: 11852

```
property ain76_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@76)

Address: 12152

```
property ain76_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN76_EF_INDEX (AIN#(0:149)_EF_INDEX@76)

Address: 9152

```
property ain76_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN76_EF_READ_A (AIN#(0:149)_EF_READ_A@76)

Address: 7152

```
property ain76_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN76_EF_READ_B (AIN#(0:149)_EF_READ_B@76)

Address: 7452

```
property ain76_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN76_EF_READ_C (AIN#(0:149)_EF_READ_C@76)

Address: 7752

```
property ain76_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN76_EF_READ_D (AIN#(0:149)_EF_READ_D@76)

Address: 8052

```
property ain76_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN76_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@76)

Address: 41076

```
property ain76_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN76_RANGE (AIN#(48:127)_RANGE@76)

Address: 40152

```
property ain76_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN76_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@76)

Address: 41576

```
property ain76_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN76_SETTLING_US (AIN#(48:127)_SETTLING_US@76)

Address: 42152

```
property ain77: float
```

Returns the voltage of the specified analog input.

Register: AIN77 (AIN#(48:127)@77)

Address: 154

```
property ain77_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN77_BINARY (AIN#(48:127)_BINARY@77)

Address: 50154

```
property ain77_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@77)

Address: 9454

```
property ain77_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@77)

Address: 9754

```
property ain77_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@77)

Address: 10054

```
property ain77_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@77)

Address: 10354

```
property ain77_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@77)

Address: 10654

```
property ain77_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@77)

Address: 10954

```
property ain77_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@77)

Address: 11254

```
property ain77_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@77)

Address: 11554

```
property ain77_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@77)

Address: 11854

```
property ain77_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@77)

Address: 12154

```
property ain77_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN77_EF_INDEX (AIN#(0:149)_EF_INDEX@77)

Address: 9154

```
property ain77_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN77_EF_READ_A (AIN#(0:149)_EF_READ_A@77)

Address: 7154

```
property ain77_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN77_EF_READ_B (AIN#(0:149)_EF_READ_B@77)

Address: 7454

```
property ain77_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN77_EF_READ_C (AIN#(0:149)_EF_READ_C@77)

Address: 7754

```
property ain77_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN77_EF_READ_D (AIN#(0:149)_EF_READ_D@77)

Address: 8054

```
property ain77_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN77_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@77)

Address: 41077

```
property ain77_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN77_RANGE (AIN#(48:127)_RANGE@77)

Address: 40154

```
property ain77_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN77_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@77)

Address: 41577

```
property ain77_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN77_SETTLING_US (AIN#(48:127)_SETTLING_US@77)

Address: 42154

```
property ain78: float
```

Returns the voltage of the specified analog input.

Register: AIN78 (AIN#(48:127)@78)

Address: 156

```
property ain78_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN78_BINARY (AIN#(48:127)_BINARY@78)

Address: 50156

```
property ain78_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@78)

Address: 9456

```
property ain78_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@78)

Address: 9756

```
property ain78_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@78)

Address: 10056

```
property ain78_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@78)

Address: 10356

```
property ain78_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@78)

Address: 10656

```
property ain78_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@78)

Address: 10956

```
property ain78_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@78)

Address: 11256

```
property ain78_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@78)

Address: 11556

```
property ain78_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@78)

Address: 11856

```
property ain78_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@78)

Address: 12156

```
property ain78_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T;

25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN78_EF_INDEX (AIN#(0:149)_EF_INDEX@78)

Address: 9156

```
property ain78_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN78_EF_READ_A (AIN#(0:149)_EF_READ_A@78)

Address: 7156

```
property ain78_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN78_EF_READ_B (AIN#(0:149)_EF_READ_B@78)

Address: 7456

```
property ain78_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN78_EF_READ_C (AIN#(0:149)_EF_READ_C@78)

Address: 7756

```
property ain78_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN78_EF_READ_D (AIN#(0:149)_EF_READ_D@78)

Address: 8056

```
property ain78_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN78_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@78)

Address: 41078

```
property ain78_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN78_RANGE (AIN#(48:127)_RANGE@78)

Address: 40156

```
property ain78_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN78_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@78)

Address: 41578

```
property ain78_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN78_SETTLING_US (AIN#(48:127)_SETTLING_US@78)

Address: 42156

```
property ain79: float
```

Returns the voltage of the specified analog input.

Register: AIN79 (AIN#(48:127)@79)

Address: 158

```
property ain79_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN79_BINARY (AIN#(48:127)_BINARY@79)

Address: 50158

```
property ain79_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@79)

Address: 9458

```
property ain79_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@79)

Address: 9758

```
property ain79_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@79)

Address: 10058

```
property ain79_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@79)

Address: 10358

```
property ain79_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@79)

Address: 10658

```
property ain79_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@79)

Address: 10958

```
property ain79_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@79)

Address: 11258

```
property ain79_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@79)

Address: 11558

```
property ain79_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@79)

Address: 11858

```
property ain79_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@79)

Address: 12158

```
property ain79_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN79_EF_INDEX (AIN#(0:149)_EF_INDEX@79)

Address: 9158

```
property ain79_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN79_EF_READ_A (AIN#(0:149)_EF_READ_A@79)

Address: 7158

```
property ain79_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN79_EF_READ_B (AIN#(0:149)_EF_READ_B@79)

Address: 7458

```
property ain79_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN79_EF_READ_C (AIN#(0:149)_EF_READ_C@79)

Address: 7758

```
property ain79_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN79_EF_READ_D (AIN#(0:149)_EF_READ_D@79)

Address: 8058

```
property ain79_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN79_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@79)

Address: 41079

```
property ain79_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN79_RANGE (AIN#(48:127)_RANGE@79)

Address: 40158

```
property ain79_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN79_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@79)

Address: 41579

```
property ain79_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN79_SETTLING_US (AIN#(48:127)_SETTLING_US@79)

Address: 42158

```
property ain7_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN7_BINARY (AIN#(0:13)_BINARY@7)

Address: 50014

```
property ain7_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@7)

Address: 9314

```
property ain7_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@7)

Address: 9614

```
property ain7_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@7)

Address: 9914

```
property ain7_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@7)

Address: 10214

```
property ain7_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@7)

Address: 10514

```
property ain7_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@7)

Address: 10814

```
property ain7_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@7)

Address: 11114

```
property ain7_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@7)

Address: 11414

```
property ain7_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@7)

Address: 11714

```
property ain7_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@7)

Address: 12014

```
property ain7_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN7_EF_INDEX (AIN#(0:149)_EF_INDEX@7)

Address: 9014

```
property ain7_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN7_EF_READ_A (AIN#(0:149)_EF_READ_A@7)

Address: 7014

```
property ain7_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN7_EF_READ_B (AIN#(0:149)_EF_READ_B@7)

Address: 7314

```
property ain7_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN7_EF_READ_C (AIN#(0:149)_EF_READ_C@7)

Address: 7614

```
property ain7_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN7_EF_READ_D (AIN#(0:149)_EF_READ_D@7)

Address: 7914

```
property ain7_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN7_NEGATIVE_CH (AIN#(0:13)_NEGATIVE_CH@7)

Address: 41007

```
property ain7_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN7_RANGE (AIN#(0:13)_RANGE@7)

Address: 40014

```
property ain7_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN7_RESOLUTION_INDEX (AIN#(0:13)_RESOLUTION_INDEX@7)

Address: 41507

```
property ain7_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN7_SETTLING_US (AIN#(0:13)_SETTLING_US@7)

Address: 42014

```
property ain8: float
```

Returns the voltage of the specified analog input.

Register: AIN8 (AIN#(0:13)@8)

Address: 16

```
property ain80: float
```

Returns the voltage of the specified analog input.

Register: AIN80 (AIN#(48:127)@80)

Address: 160

```
property ain80_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN80_BINARY (AIN#(48:127)_BINARY@80)

Address: 50160

```
property ain80_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@80)

Address: 9460

```
property ain80_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@80)

Address: 9760

```
property ain80_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@80)

Address: 10060

```
property ain80_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@80)

Address: 10360

```
property ain80_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@80)

Address: 10660

```
property ain80_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@80)

Address: 10960

```
property ain80_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@80)

Address: 11260

```
property ain80_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@80)

Address: 11560

```
property ain80_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@80)

Address: 11860

```
property ain80_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@80)

Address: 12160

```
property ain80_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN80_EF_INDEX (AIN#(0:149)_EF_INDEX@80)

Address: 9160

```
property ain80_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN80_EF_READ_A (AIN#(0:149)_EF_READ_A@80)

Address: 7160

```
property ain80_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN80_EF_READ_B (AIN#(0:149)_EF_READ_B@80)

Address: 7460

```
property ain80_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN80_EF_READ_C (AIN#(0:149)_EF_READ_C@80)

Address: 7760

```
property ain80_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN80_EF_READ_D (AIN#(0:149)_EF_READ_D@80)

Address: 8060

```
property ain80_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN80_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@80)

Address: 41080

```
property ain80_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN80_RANGE (AIN#(48:127)_RANGE@80)

Address: 40160

```
property ain80_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN80_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@80)

Address: 41580

```
property ain80_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN80_SETTLING_US (AIN#(48:127)_SETTLING_US@80)

Address: 42160

```
property ain81: float
```

Returns the voltage of the specified analog input.

Register: AIN81 (AIN#(48:127)@81)

Address: 162

```
property ain81_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN81_BINARY (AIN#(48:127)_BINARY@81)

Address: 50162

```
property ain81_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@81)

Address: 9462

```
property ain81_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@81)

Address: 9762

```
property ain81_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@81)

Address: 10062

```
property ain81_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@81)

Address: 10362

```
property ain81_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@81)

Address: 10662

```
property ain81_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@81)

Address: 10962

```
property ain81_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@81)

Address: 11262

```
property ain81_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@81)

Address: 11562

```
property ain81_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@81)

Address: 11862

```
property ain81_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@81)

Address: 12162

```
property ain81_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN81_EF_INDEX (AIN#(0:149)_EF_INDEX@81)

Address: 9162

```
property ain81_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN81_EF_READ_A (AIN#(0:149)_EF_READ_A@81)

Address: 7162

```
property ain81_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN81_EF_READ_B (AIN#(0:149)_EF_READ_B@81)

Address: 7462

```
property ain81_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN81_EF_READ_C (AIN#(0:149)_EF_READ_C@81)

Address: 7762

```
property ain81_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN81_EF_READ_D (AIN#(0:149)_EF_READ_D@81)

Address: 8062

```
property ain81_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN81_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@81)

Address: 41081

```
property ain81_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN81_RANGE (AIN#(48:127)_RANGE@81)

Address: 40162

```
property ain81_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN81_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@81)

Address: 41581

```
property ain81_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN81_SETTLING_US (AIN#(48:127)_SETTLING_US@81)

Address: 42162

```
property ain82: float
```

Returns the voltage of the specified analog input.

Register: AIN82 (AIN#(48:127)@82)

Address: 164

```
property ain82_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN82_BINARY (AIN#(48:127)_BINARY@82)

Address: 50164

```
property ain82_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@82)

Address: 9464

```
property ain82_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@82)

Address: 9764

```
property ain82_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@82)

Address: 10064

```
property ain82_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@82)

Address: 10364

```
property ain82_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@82)

Address: 10664

```
property ain82_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@82)

Address: 10964

```
property ain82_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@82)

Address: 11264

```
property ain82_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@82)

Address: 11564

```
property ain82_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@82)

Address: 11864

```
property ain82_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@82)

Address: 12164

```
property ain82_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN82_EF_INDEX (AIN#(0:149)_EF_INDEX@82)

Address: 9164

```
property ain82_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN82_EF_READ_A (AIN#(0:149)_EF_READ_A@82)

Address: 7164

```
property ain82_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN82_EF_READ_B (AIN#(0:149)_EF_READ_B@82)

Address: 7464

```
property ain82_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN82_EF_READ_C (AIN#(0:149)_EF_READ_C@82)

Address: 7764

```
property ain82_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN82_EF_READ_D (AIN#(0:149)_EF_READ_D@82)

Address: 8064

```
property ain82_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN82_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@82)

Address: 41082

```
property ain82_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN82_RANGE (AIN#(48:127)_RANGE@82)

Address: 40164

```
property ain82_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN82_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@82)

Address: 41582

```
property ain82_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN82_SETTLING_US (AIN#(48:127)_SETTLING_US@82)

Address: 42164

```
property ain83: float
```

Returns the voltage of the specified analog input.

Register: AIN83 (AIN#(48:127)@83)

Address: 166

```
property ain83_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN83_BINARY (AIN#(48:127)_BINARY@83)

Address: 50166

```
property ain83_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@83)

Address: 9466

```
property ain83_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@83)

Address: 9766

```
property ain83_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@83)

Address: 10066

```
property ain83_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@83)

Address: 10366

```
property ain83_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@83)

Address: 10666

```
property ain83_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@83)

Address: 10966

```
property ain83_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@83)

Address: 11266

```
property ain83_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@83)

Address: 11566

```
property ain83_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@83)

Address: 11866

```
property ain83_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@83)

Address: 12166

```
property ain83_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN83_EF_INDEX (AIN#(0:149)_EF_INDEX@83)

Address: 9166

```
property ain83_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN83_EF_READ_A (AIN#(0:149)_EF_READ_A@83)

Address: 7166

```
property ain83_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN83_EF_READ_B (AIN#(0:149)_EF_READ_B@83)

Address: 7466

```
property ain83_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN83_EF_READ_C (AIN#(0:149)_EF_READ_C@83)

Address: 7766

```
property ain83_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN83_EF_READ_D (AIN#(0:149)_EF_READ_D@83)

Address: 8066

```
property ain83_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN83_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@83)

Address: 41083

```
property ain83_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN83_RANGE (AIN#(48:127)_RANGE@83)

Address: 40166

```
property ain83_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN83_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@83)

Address: 41583

```
property ain83_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN83_SETTLING_US (AIN#(48:127)_SETTLING_US@83)

Address: 42166

```
property ain84: float
```

Returns the voltage of the specified analog input.

Register: AIN84 (AIN#(48:127)@84)

Address: 168

```
property ain84_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN84_BINARY (AIN#(48:127)_BINARY@84)

Address: 50168

```
property ain84_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@84)

Address: 9468

```
property ain84_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@84)

Address: 9768

```
property ain84_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@84)

Address: 10068

```
property ain84_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@84)

Address: 10368

```
property ain84_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@84)

Address: 10668

```
property ain84_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@84)

Address: 10968

```
property ain84_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@84)

Address: 11268

```
property ain84_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@84)

Address: 11568

```
property ain84_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@84)

Address: 11868

```
property ain84_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@84)

Address: 12168

```
property ain84_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN84_EF_INDEX (AIN#(0:149)_EF_INDEX@84)

Address: 9168

```
property ain84_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN84_EF_READ_A (AIN#(0:149)_EF_READ_A@84)

Address: 7168

```
property ain84_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN84_EF_READ_B (AIN#(0:149)_EF_READ_B@84)

Address: 7468

```
property ain84_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN84_EF_READ_C (AIN#(0:149)_EF_READ_C@84)

Address: 7768

```
property ain84_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN84_EF_READ_D (AIN#(0:149)_EF_READ_D@84)

Address: 8068

```
property ain84_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN84_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@84)

Address: 41084

```
property ain84_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN84_RANGE (AIN#(48:127)_RANGE@84)

Address: 40168

```
property ain84_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN84_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@84)

Address: 41584

```
property ain84_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN84_SETTLING_US (AIN#(48:127)_SETTLING_US@84)

Address: 42168

```
property ain85: float
```

Returns the voltage of the specified analog input.

Register: AIN85 (AIN#(48:127)@85)

Address: 170

```
property ain85_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN85_BINARY (AIN#(48:127)_BINARY@85)

Address: 50170

```
property ain85_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@85)

Address: 9470

```
property ain85_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@85)

Address: 9770

```
property ain85_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@85)

Address: 10070

```
property ain85_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@85)

Address: 10370

```
property ain85_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@85)

Address: 10670

```
property ain85_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@85)

Address: 10970

```
property ain85_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@85)

Address: 11270

```
property ain85_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@85)

Address: 11570

```
property ain85_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@85)

Address: 11870

```
property ain85_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@85)

Address: 12170

```
property ain85_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN85_EF_INDEX (AIN#(0:149)_EF_INDEX@85)

Address: 9170

```
property ain85_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN85_EF_READ_A (AIN#(0:149)_EF_READ_A@85)

Address: 7170

```
property ain85_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN85_EF_READ_B (AIN#(0:149)_EF_READ_B@85)

Address: 7470

```
property ain85_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN85_EF_READ_C (AIN#(0:149)_EF_READ_C@85)

Address: 7770

```
property ain85_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN85_EF_READ_D (AIN#(0:149)_EF_READ_D@85)

Address: 8070

```
property ain85_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN85_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@85)

Address: 41085

```
property ain85_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN85_RANGE (AIN#(48:127)_RANGE@85)

Address: 40170

```
property ain85_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN85_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@85)

Address: 41585

```
property ain85_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN85_SETTLING_US (AIN#(48:127)_SETTLING_US@85)

Address: 42170

```
property ain86: float
```

Returns the voltage of the specified analog input.

Register: AIN86 (AIN#(48:127)@86)

Address: 172

```
property ain86_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN86_BINARY (AIN#(48:127)_BINARY@86)

Address: 50172

```
property ain86_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@86)

Address: 9472

```
property ain86_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@86)

Address: 9772

```
property ain86_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@86)

Address: 10072

```
property ain86_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@86)

Address: 10372

```
property ain86_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@86)

Address: 10672

```
property ain86_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@86)

Address: 10972

```
property ain86_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@86)

Address: 11272

```
property ain86_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@86)

Address: 11572

```
property ain86_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@86)

Address: 11872

```
property ain86_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@86)

Address: 12172

```
property ain86_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN86_EF_INDEX (AIN#(0:149)_EF_INDEX@86)

Address: 9172

```
property ain86_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN86_EF_READ_A (AIN#(0:149)_EF_READ_A@86)

Address: 7172

```
property ain86_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN86_EF_READ_B (AIN#(0:149)_EF_READ_B@86)

Address: 7472

```
property ain86_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN86_EF_READ_C (AIN#(0:149)_EF_READ_C@86)

Address: 7772

```
property ain86_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN86_EF_READ_D (AIN#(0:149)_EF_READ_D@86)

Address: 8072

```
property ain86_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN86_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@86)

Address: 41086

```
property ain86_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN86_RANGE (AIN#(48:127)_RANGE@86)

Address: 40172

```
property ain86_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN86_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@86)

Address: 41586

```
property ain86_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN86_SETTLING_US (AIN#(48:127)_SETTLING_US@86)

Address: 42172

```
property ain87: float
```

Returns the voltage of the specified analog input.

Register: AIN87 (AIN#(48:127)@87)

Address: 174

```
property ain87_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN87_BINARY (AIN#(48:127)_BINARY@87)

Address: 50174

```
property ain87_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@87)

Address: 9474

```
property ain87_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@87)

Address: 9774

```
property ain87_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@87)

Address: 10074

```
property ain87_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@87)

Address: 10374

```
property ain87_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@87)

Address: 10674

```
property ain87_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@87)

Address: 10974

```
property ain87_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@87)

Address: 11274

```
property ain87_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@87)

Address: 11574

```
property ain87_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@87)

Address: 11874

```
property ain87_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@87)

Address: 12174

```
property ain87_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN87_EF_INDEX (AIN#(0:149)_EF_INDEX@87)

Address: 9174

```
property ain87_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN87_EF_READ_A (AIN#(0:149)_EF_READ_A@87)

Address: 7174

```
property ain87_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN87_EF_READ_B (AIN#(0:149)_EF_READ_B@87)

Address: 7474

```
property ain87_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN87_EF_READ_C (AIN#(0:149)_EF_READ_C@87)

Address: 7774

```
property ain87_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN87_EF_READ_D (AIN#(0:149)_EF_READ_D@87)

Address: 8074

```
property ain87_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN87_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@87)

Address: 41087

```
property ain87_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN87_RANGE (AIN#(48:127)_RANGE@87)

Address: 40174

```
property ain87_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN87_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@87)

Address: 41587

```
property ain87_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN87_SETTLING_US (AIN#(48:127)_SETTLING_US@87)

Address: 42174

```
property ain88: float
```

Returns the voltage of the specified analog input.

Register: AIN88 (AIN#(48:127)@88)

Address: 176

```
property ain88_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN88_BINARY (AIN#(48:127)_BINARY@88)

Address: 50176

```
property ain88_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@88)

Address: 9476

```
property ain88_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@88)

Address: 9776

```
property ain88_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@88)

Address: 10076

```
property ain88_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@88)

Address: 10376

```
property ain88_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@88)

Address: 10676

```
property ain88_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@88)

Address: 10976

```
property ain88_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@88)

Address: 11276

```
property ain88_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@88)

Address: 11576

```
property ain88_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@88)

Address: 11876

```
property ain88_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@88)

Address: 12176

```
property ain88_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T;

25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN88_EF_INDEX (AIN#(0:149)_EF_INDEX@88)

Address: 9176

```
property ain88_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN88_EF_READ_A (AIN#(0:149)_EF_READ_A@88)

Address: 7176

```
property ain88_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN88_EF_READ_B (AIN#(0:149)_EF_READ_B@88)

Address: 7476

```
property ain88_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN88_EF_READ_C (AIN#(0:149)_EF_READ_C@88)

Address: 7776

```
property ain88_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN88_EF_READ_D (AIN#(0:149)_EF_READ_D@88)

Address: 8076

```
property ain88_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN88_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@88)

Address: 41088

```
property ain88_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN88_RANGE (AIN#(48:127)_RANGE@88)

Address: 40176

```
property ain88_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN88_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@88)

Address: 41588

```
property ain88_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN88_SETTLING_US (AIN#(48:127)_SETTLING_US@88)

Address: 42176

```
property ain89: float
```

Returns the voltage of the specified analog input.

Register: AIN89 (AIN#(48:127)@89)

Address: 178

```
property ain89_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN89_BINARY (AIN#(48:127)_BINARY@89)

Address: 50178

```
property ain89_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@89)

Address: 9478

```
property ain89_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@89)

Address: 9778

```
property ain89_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@89)

Address: 10078

```
property ain89_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@89)

Address: 10378

```
property ain89_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@89)

Address: 10678

```
property ain89_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@89)

Address: 10978

```
property ain89_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@89)

Address: 11278

```
property ain89_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@89)

Address: 11578

```
property ain89_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@89)

Address: 11878

```
property ain89_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@89)

Address: 12178

```
property ain89_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN89_EF_INDEX (AIN#(0:149)_EF_INDEX@89)

Address: 9178

```
property ain89_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN89_EF_READ_A (AIN#(0:149)_EF_READ_A@89)

Address: 7178

```
property ain89_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN89_EF_READ_B (AIN#(0:149)_EF_READ_B@89)

Address: 7478

```
property ain89_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN89_EF_READ_C (AIN#(0:149)_EF_READ_C@89)

Address: 7778

```
property ain89_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN89_EF_READ_D (AIN#(0:149)_EF_READ_D@89)

Address: 8078

```
property ain89_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN89_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@89)

Address: 41089

```
property ain89_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN89_RANGE (AIN#(48:127)_RANGE@89)

Address: 40178

```
property ain89_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN89_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@89)

Address: 41589

```
property ain89_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN89_SETTLING_US (AIN#(48:127)_SETTLING_US@89)

Address: 42178

```
property ain8_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN8_BINARY (AIN#(0:13)_BINARY@8)

Address: 50016

```
property ain8_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@8)

Address: 9316

```
property ain8_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@8)

Address: 9616

```
property ain8_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@8)

Address: 9916

```
property ain8_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@8)

Address: 10216

```
property ain8_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@8)

Address: 10516

```
property ain8_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@8)

Address: 10816

```
property ain8_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@8)

Address: 11116

```
property ain8_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@8)

Address: 11416

```
property ain8_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@8)

Address: 11716

```
property ain8_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@8)

Address: 12016

```
property ain8_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN8_EF_INDEX (AIN#(0:149)_EF_INDEX@8)

Address: 9016

```
property ain8_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN8_EF_READ_A (AIN#(0:149)_EF_READ_A@8)

Address: 7016

```
property ain8_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN8_EF_READ_B (AIN#(0:149)_EF_READ_B@8)

Address: 7316

```
property ain8_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN8_EF_READ_C (AIN#(0:149)_EF_READ_C@8)

Address: 7616

```
property ain8_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN8_EF_READ_D (AIN#(0:149)_EF_READ_D@8)

Address: 7916

```
property ain8_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN8_NEGATIVE_CH (AIN#(0:13)_NEGATIVE_CH@8)

Address: 41008

```
property ain8_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN8_RANGE (AIN#(0:13)_RANGE@8)

Address: 40016

```
property ain8_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN8_RESOLUTION_INDEX (AIN#(0:13)_RESOLUTION_INDEX@8)

Address: 41508

```
property ain8_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN8_SETTLING_US (AIN#(0:13)_SETTLING_US@8)

Address: 42016

```
property ain9: float
```

Returns the voltage of the specified analog input.

Register: AIN9 (AIN#(0:13)@9)

Address: 18

```
property ain90: float
```

Returns the voltage of the specified analog input.

Register: AIN90 (AIN#(48:127)@90)

Address: 180

```
property ain90_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN90_BINARY (AIN#(48:127)_BINARY@90)

Address: 50180

```
property ain90_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@90)

Address: 9480

```
property ain90_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@90)

Address: 9780

```
property ain90_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@90)

Address: 10080

```
property ain90_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@90)

Address: 10380

```
property ain90_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@90)

Address: 10680

```
property ain90_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@90)

Address: 10980

```
property ain90_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@90)

Address: 11280

```
property ain90_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@90)

Address: 11580

```
property ain90_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@90)

Address: 11880

```
property ain90_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@90)

Address: 12180

```
property ain90_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN90_EF_INDEX (AIN#(0:149)_EF_INDEX@90)

Address: 9180

```
property ain90_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN90_EF_READ_A (AIN#(0:149)_EF_READ_A@90)

Address: 7180

```
property ain90_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN90_EF_READ_B (AIN#(0:149)_EF_READ_B@90)

Address: 7480

```
property ain90_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN90_EF_READ_C (AIN#(0:149)_EF_READ_C@90)

Address: 7780

```
property ain90_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN90_EF_READ_D (AIN#(0:149)_EF_READ_D@90)

Address: 8080

```
property ain90_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN90_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@90)

Address: 41090

```
property ain90_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN90_RANGE (AIN#(48:127)_RANGE@90)

Address: 40180

```
property ain90_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN90_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@90)

Address: 41590

```
property ain90_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN90_SETTLING_US (AIN#(48:127)_SETTLING_US@90)

Address: 42180

```
property ain91: float
```

Returns the voltage of the specified analog input.

Register: AIN91 (AIN#(48:127)@91)

Address: 182

```
property ain91_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN91_BINARY (AIN#(48:127)_BINARY@91)

Address: 50182

```
property ain91_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@91)

Address: 9482

```
property ain91_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@91)

Address: 9782

```
property ain91_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@91)

Address: 10082

```
property ain91_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@91)

Address: 10382

```
property ain91_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@91)

Address: 10682

```
property ain91_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@91)

Address: 10982

```
property ain91_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@91)

Address: 11282

```
property ain91_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@91)

Address: 11582

```
property ain91_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@91)

Address: 11882

```
property ain91_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@91)

Address: 12182

```
property ain91_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN91_EF_INDEX (AIN#(0:149)_EF_INDEX@91)

Address: 9182

```
property ain91_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN91_EF_READ_A (AIN#(0:149)_EF_READ_A@91)

Address: 7182

```
property ain91_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN91_EF_READ_B (AIN#(0:149)_EF_READ_B@91)

Address: 7482

```
property ain91_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN91_EF_READ_C (AIN#(0:149)_EF_READ_C@91)

Address: 7782

```
property ain91_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN91_EF_READ_D (AIN#(0:149)_EF_READ_D@91)

Address: 8082

```
property ain91_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN91_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@91)

Address: 41091

```
property ain91_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN91_RANGE (AIN#(48:127)_RANGE@91)

Address: 40182

```
property ain91_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN91_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@91)

Address: 41591

```
property ain91_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN91_SETTLING_US (AIN#(48:127)_SETTLING_US@91)

Address: 42182

```
property ain92: float
```

Returns the voltage of the specified analog input.

Register: AIN92 (AIN#(48:127)@92)

Address: 184

```
property ain92_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN92_BINARY (AIN#(48:127)_BINARY@92)

Address: 50184

```
property ain92_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@92)

Address: 9484

```
property ain92_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@92)

Address: 9784

```
property ain92_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@92)

Address: 10084

```
property ain92_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@92)

Address: 10384

```
property ain92_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@92)

Address: 10684

```
property ain92_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@92)

Address: 10984

```
property ain92_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@92)

Address: 11284

```
property ain92_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@92)

Address: 11584

```
property ain92_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@92)

Address: 11884

```
property ain92_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@92)

Address: 12184

```
property ain92_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN92_EF_INDEX (AIN#(0:149)_EF_INDEX@92)

Address: 9184

```
property ain92_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN92_EF_READ_A (AIN#(0:149)_EF_READ_A@92)

Address: 7184

```
property ain92_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN92_EF_READ_B (AIN#(0:149)_EF_READ_B@92)

Address: 7484

```
property ain92_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN92_EF_READ_C (AIN#(0:149)_EF_READ_C@92)

Address: 7784

```
property ain92_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN92_EF_READ_D (AIN#(0:149)_EF_READ_D@92)

Address: 8084

```
property ain92_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN92_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@92)

Address: 41092

```
property ain92_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN92_RANGE (AIN#(48:127)_RANGE@92)

Address: 40184

```
property ain92_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN92_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@92)

Address: 41592

```
property ain92_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN92_SETTLING_US (AIN#(48:127)_SETTLING_US@92)

Address: 42184

```
property ain93: float
```

Returns the voltage of the specified analog input.

Register: AIN93 (AIN#(48:127)@93)

Address: 186

```
property ain93_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN93_BINARY (AIN#(48:127)_BINARY@93)

Address: 50186

```
property ain93_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@93)

Address: 9486

```
property ain93_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@93)

Address: 9786

```
property ain93_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@93)

Address: 10086

```
property ain93_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@93)

Address: 10386

```
property ain93_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@93)

Address: 10686

```
property ain93_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@93)

Address: 10986

```
property ain93_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@93)

Address: 11286

```
property ain93_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@93)

Address: 11586

```
property ain93_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@93)

Address: 11886

```
property ain93_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@93)

Address: 12186

```
property ain93_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN93_EF_INDEX (AIN#(0:149)_EF_INDEX@93)

Address: 9186

```
property ain93_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN93_EF_READ_A (AIN#(0:149)_EF_READ_A@93)

Address: 7186

```
property ain93_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN93_EF_READ_B (AIN#(0:149)_EF_READ_B@93)

Address: 7486

```
property ain93_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN93_EF_READ_C (AIN#(0:149)_EF_READ_C@93)

Address: 7786

```
property ain93_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN93_EF_READ_D (AIN#(0:149)_EF_READ_D@93)

Address: 8086

```
property ain93_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN93_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@93)

Address: 41093

```
property ain93_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN93_RANGE (AIN#(48:127)_RANGE@93)

Address: 40186

```
property ain93_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN93_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@93)

Address: 41593

```
property ain93_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN93_SETTLING_US (AIN#(48:127)_SETTLING_US@93)

Address: 42186

```
property ain94: float
```

Returns the voltage of the specified analog input.

Register: AIN94 (AIN#(48:127)@94)

Address: 188

```
property ain94_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN94_BINARY (AIN#(48:127)_BINARY@94)

Address: 50188

```
property ain94_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@94)

Address: 9488

```
property ain94_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@94)

Address: 9788

```
property ain94_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@94)

Address: 10088

```
property ain94_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@94)

Address: 10388

```
property ain94_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@94)

Address: 10688

```
property ain94_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@94)

Address: 10988

```
property ain94_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@94)

Address: 11288

```
property ain94_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@94)

Address: 11588

```
property ain94_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@94)

Address: 11888

```
property ain94_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@94)

Address: 12188

```
property ain94_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN94_EF_INDEX (AIN#(0:149)_EF_INDEX@94)

Address: 9188

```
property ain94_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN94_EF_READ_A (AIN#(0:149)_EF_READ_A@94)

Address: 7188

```
property ain94_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN94_EF_READ_B (AIN#(0:149)_EF_READ_B@94)

Address: 7488

```
property ain94_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN94_EF_READ_C (AIN#(0:149)_EF_READ_C@94)

Address: 7788

```
property ain94_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN94_EF_READ_D (AIN#(0:149)_EF_READ_D@94)

Address: 8088

```
property ain94_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN94_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@94)

Address: 41094

```
property ain94_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN94_RANGE (AIN#(48:127)_RANGE@94)

Address: 40188

```
property ain94_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN94_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@94)

Address: 41594

```
property ain94_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN94_SETTLING_US (AIN#(48:127)_SETTLING_US@94)

Address: 42188

```
property ain95: float
```

Returns the voltage of the specified analog input.

Register: AIN95 (AIN#(48:127)@95)

Address: 190

```
property ain95_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN95_BINARY (AIN#(48:127)_BINARY@95)

Address: 50190

```
property ain95_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@95)

Address: 9490

```
property ain95_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@95)

Address: 9790

```
property ain95_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@95)

Address: 10090

```
property ain95_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@95)

Address: 10390

```
property ain95_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@95)

Address: 10690

```
property ain95_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@95)

Address: 10990

```
property ain95_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@95)

Address: 11290

```
property ain95_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@95)

Address: 11590

```
property ain95_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@95)

Address: 11890

```
property ain95_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@95)

Address: 12190

```
property ain95_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN95_EF_INDEX (AIN#(0:149)_EF_INDEX@95)

Address: 9190

```
property ain95_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN95_EF_READ_A (AIN#(0:149)_EF_READ_A@95)

Address: 7190

```
property ain95_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN95_EF_READ_B (AIN#(0:149)_EF_READ_B@95)

Address: 7490

```
property ain95_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN95_EF_READ_C (AIN#(0:149)_EF_READ_C@95)

Address: 7790

```
property ain95_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN95_EF_READ_D (AIN#(0:149)_EF_READ_D@95)

Address: 8090

```
property ain95_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN95_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@95)

Address: 41095

```
property ain95_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN95_RANGE (AIN#(48:127)_RANGE@95)

Address: 40190

```
property ain95_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN95_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@95)

Address: 41595

```
property ain95_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN95_SETTLING_US (AIN#(48:127)_SETTLING_US@95)

Address: 42190

```
property ain96: float
```

Returns the voltage of the specified analog input.

Register: AIN96 (AIN#(48:127)@96)

Address: 192

```
property ain96_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN96_BINARY (AIN#(48:127)_BINARY@96)

Address: 50192

```
property ain96_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@96)

Address: 9492

```
property ain96_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@96)

Address: 9792

```
property ain96_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@96)

Address: 10092

```
property ain96_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@96)

Address: 10392

```
property ain96_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@96)

Address: 10692

```
property ain96_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@96)

Address: 10992

```
property ain96_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@96)

Address: 11292

```
property ain96_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@96)

Address: 11592

```
property ain96_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@96)

Address: 11892

```
property ain96_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@96)

Address: 12192

```
property ain96_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN96_EF_INDEX (AIN#(0:149)_EF_INDEX@96)

Address: 9192

```
property ain96_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN96_EF_READ_A (AIN#(0:149)_EF_READ_A@96)

Address: 7192

```
property ain96_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN96_EF_READ_B (AIN#(0:149)_EF_READ_B@96)

Address: 7492

```
property ain96_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN96_EF_READ_C (AIN#(0:149)_EF_READ_C@96)

Address: 7792

```
property ain96_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN96_EF_READ_D (AIN#(0:149)_EF_READ_D@96)

Address: 8092

```
property ain96_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN96_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@96)

Address: 41096

```
property ain96_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN96_RANGE (AIN#(48:127)_RANGE@96)

Address: 40192

```
property ain96_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN96_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@96)

Address: 41596

```
property ain96_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN96_SETTLING_US (AIN#(48:127)_SETTLING_US@96)

Address: 42192

```
property ain97: float
```

Returns the voltage of the specified analog input.

Register: AIN97 (AIN#(48:127)@97)

Address: 194

```
property ain97_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN97_BINARY (AIN#(48:127)_BINARY@97)

Address: 50194

```
property ain97_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@97)

Address: 9494

```
property ain97_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@97)

Address: 9794

```
property ain97_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@97)

Address: 10094

```
property ain97_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@97)

Address: 10394

```
property ain97_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@97)

Address: 10694

```
property ain97_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@97)

Address: 10994

```
property ain97_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@97)

Address: 11294

```
property ain97_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@97)

Address: 11594

```
property ain97_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@97)

Address: 11894

```
property ain97_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@97)

Address: 12194

```
property ain97_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN97_EF_INDEX (AIN#(0:149)_EF_INDEX@97)

Address: 9194

```
property ain97_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN97_EF_READ_A (AIN#(0:149)_EF_READ_A@97)

Address: 7194

```
property ain97_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN97_EF_READ_B (AIN#(0:149)_EF_READ_B@97)

Address: 7494

```
property ain97_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN97_EF_READ_C (AIN#(0:149)_EF_READ_C@97)

Address: 7794

```
property ain97_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN97_EF_READ_D (AIN#(0:149)_EF_READ_D@97)

Address: 8094

```
property ain97_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN97_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@97)

Address: 41097

```
property ain97_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN97_RANGE (AIN#(48:127)_RANGE@97)

Address: 40194

```
property ain97_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN97_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@97)

Address: 41597

```
property ain97_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN97_SETTLING_US (AIN#(48:127)_SETTLING_US@97)

Address: 42194

```
property ain98: float
```

Returns the voltage of the specified analog input.

Register: AIN98 (AIN#(48:127)@98)

Address: 196

```
property ain98_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN98_BINARY (AIN#(48:127)_BINARY@98)

Address: 50196

```
property ain98_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@98)

Address: 9496

```
property ain98_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@98)

Address: 9796

```
property ain98_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@98)

Address: 10096

```
property ain98_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@98)

Address: 10396

```
property ain98_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@98)

Address: 10696

```
property ain98_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@98)

Address: 10996

```
property ain98_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@98)

Address: 11296

```
property ain98_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@98)

Address: 11596

```
property ain98_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@98)

Address: 11896

```
property ain98_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@98)

Address: 12196

```
property ain98_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T;

25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN98_EF_INDEX (AIN#(0:149)_EF_INDEX@98)

Address: 9196

```
property ain98_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN98_EF_READ_A (AIN#(0:149)_EF_READ_A@98)

Address: 7196

```
property ain98_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN98_EF_READ_B (AIN#(0:149)_EF_READ_B@98)

Address: 7496

```
property ain98_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN98_EF_READ_C (AIN#(0:149)_EF_READ_C@98)

Address: 7796

```
property ain98_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN98_EF_READ_D (AIN#(0:149)_EF_READ_D@98)

Address: 8096

```
property ain98_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN98_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@98)

Address: 41098

```
property ain98_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN98_RANGE (AIN#(48:127)_RANGE@98)

Address: 40196

```
property ain98_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN98_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@98)

Address: 41598

```
property ain98_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN98_SETTLING_US (AIN#(48:127)_SETTLING_US@98)

Address: 42196

```
property ain99: float
```

Returns the voltage of the specified analog input.

Register: AIN99 (AIN#(48:127)@99)

Address: 198

```
property ain99_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN99_BINARY (AIN#(48:127)_BINARY@99)

Address: 50198

```
property ain99_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@99)

Address: 9498

```
property ain99_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@99)

Address: 9798

```
property ain99_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@99)

Address: 10098

```
property ain99_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@99)

Address: 10398

```
property ain99_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@99)

Address: 10698

```
property ain99_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@99)

Address: 10998

```
property ain99_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@99)

Address: 11298

```
property ain99_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@99)

Address: 11598

```
property ain99_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@99)

Address: 11898

```
property ain99_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@99)

Address: 12198

```
property ain99_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN99_EF_INDEX (AIN#(0:149)_EF_INDEX@99)

Address: 9198

```
property ain99_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN99_EF_READ_A (AIN#(0:149)_EF_READ_A@99)

Address: 7198

```
property ain99_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN99_EF_READ_B (AIN#(0:149)_EF_READ_B@99)

Address: 7498

```
property ain99_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN99_EF_READ_C (AIN#(0:149)_EF_READ_C@99)

Address: 7798

```
property ain99_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN99_EF_READ_D (AIN#(0:149)_EF_READ_D@99)

Address: 8098

```
property ain99_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN99_NEGATIVE_CH (AIN#(48:127)_NEGATIVE_CH@99)

Address: 41099

```
property ain99_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN99_RANGE (AIN#(48:127)_RANGE@99)

Address: 40198

```
property ain99_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN99_RESOLUTION_INDEX (AIN#(48:127)_RESOLUTION_INDEX@99)

Address: 41599

```
property ain99_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN99_SETTLING_US (AIN#(48:127)_SETTLING_US@99)

Address: 42198

```
property ain9_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN9_BINARY (AIN#(0:13)_BINARY@9)

Address: 50018

```
property ain9_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@9)

Address: 9318

```
property ain9_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@9)

Address: 9618

```
property ain9_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@9)

Address: 9918

```
property ain9_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@9)

Address: 10218

```
property ain9_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@9)

Address: 10518

```
property ain9_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@9)

Address: 10818

```
property ain9_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@9)

Address: 11118

```
property ain9_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@9)

Address: 11418

```
property ain9_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@9)

Address: 11718

```
property ain9_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@9)

Address: 12018

```
property ain9_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN9_EF_INDEX (AIN#(0:149)_EF_INDEX@9)

Address: 9018

```
property ain9_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN9_EF_READ_A (AIN#(0:149)_EF_READ_A@9)

Address: 7018

```
property ain9_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN9_EF_READ_B (AIN#(0:149)_EF_READ_B@9)

Address: 7318

```
property ain9_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN9_EF_READ_C (AIN#(0:149)_EF_READ_C@9)

Address: 7618

```
property ain9_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN9_EF_READ_D (AIN#(0:149)_EF_READ_D@9)

Address: 7918

```
property ain9_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN9_NEGATIVE_CH (AIN#(0:13)_NEGATIVE_CH@9)

Address: 41009

```
property ain9_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN9_RANGE (AIN#(0:13)_RANGE@9)

Address: 40018

```
property ain9_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN9_RESOLUTION_INDEX (AIN#(0:13)_RESOLUTION_INDEX@9)

Address: 41509

```
property ain9_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN9_SETTLING_US (AIN#(0:13)_SETTLING_US@9)

Address: 42018

```
property ain_all_ef_index: int
```

Write 0 to deactivate AIN_EF on all AINs. No other values may be written to this register. Reads will return the AIN_EF index if all 128 AINs are set to the same value. If values are not the same returns 0xFFFF (65535).

Register: AIN_ALL_EF_INDEX

Address: 43906

```
property ain_all_negative_ch: int
```

A write to this global parameter affects all AIN. Writing 1 will set all AINs to differential. Writing 199 will set all AINs to single-ended. A read will return 1 if all AINs are set to differential and 199 if all AINs are set to single-ended. If AIN configurations are not consistent 0xFFFF will be returned.

Register: AIN_ALL_NEGATIVE_CH

Address: 43902

```
property ain_all_range: float
```

A write to this global parameter affects all AIN. A read will return the correct setting if all channels are set the same, but otherwise will return -9999.

Register: AIN_ALL_RANGE

Address: 43900

```
property ain_all_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times. A write to this global parameter affects all AIN. A read will return the correct setting if all channels are set the same, but otherwise will return 0xFFFF.

Register: AIN_ALL_RESOLUTION_INDEX

Address: 43903

```
property ain_all_settling_us: float
```

Settling time for command-response and AIN-EF readings. A write to this global parameter affects all AIN. A read will return the correct setting if all channels are set the same, but otherwise will return -9999. Max is 50,000 us.

Register: AIN_ALL_SETTLING_US

Address: 43904

```
property asynch_baud: int
```

The symbol rate that will be used for communication. 9600 is typical. Up to 38400 works, but heavily loads the T7's processor.

Register: ASYNCH_BAUD

Address: 5420

```
property asynch_data_rx: int
```

Read received data from here.

This register is a buffer. Underrun behavior - fill with zeros.

Register: ASYNCH_DATA_RX

Address: 5495

```
property asynch_data_tx: int
```

Write data to be transmitted here. This register is a buffer.

Register: ASYNCH_DATA_TX

Address: 5490

```
property asynch_enable: int
```

1 = Turn on Asynch. Configures timing hardware, DIO lines and allocates the receiving buffer.

Register: ASYNCH_ENABLE

Address: 5400

```
property asynch_num_bytes_rx: int
```

The number of data bytes that have been received.

Register: ASYNCH_NUM_BYTES_RX

Address: 5435

```
property asynch_num_bytes_tx: int
```

The number of bytes to be transmitted after writing to GO. Max is 256.

Register: ASYNCH_NUM_BYTES_TX

Address: 5440

```
property asynch_num_data_bits: int
```

The number of data bits per frame. 0-8, 0=8.

Register: ASYNCH_NUM_DATA_BITS

Address: 5415

```
property asynch_num_parity_errors: int
```

The number of parity errors that have been detected. Cleared when UART is enabled. Can also be cleared by writing 0.

Register: ASYNCH_NUM_PARITY_ERRORS

Address: 5465

```
property asynch_num_stop_bits: int
```

The number of stop bits. Values: 0 = zero stop bits, 1 = one stop bit, 2 = two stop bits.

Register: ASYNCH_NUM_STOP_BITS

Address: 5455

```
property asynch_parity: int
```

Parity setting: 0=none, 1=odd, 2=even.

Register: ASYNCH_PARITY

Address: 5460

```
property asynch_rx_buffer_size_bytes: int
```

Number of bytes to use for the receiving buffer. Max is 2048. 0 = 200.

Register: ASYNCH_RX_BUFFER_SIZE_BYTES

Address: 5430

```
property asynch_rx_dionum: int
```

The DIO line that will receive data. (RX)

Register: ASYNCH_RX_DIONUM

Address: 5405

```
property asynch_tx_dionum: int
```

The DIO line that will transmit data. (TX)

Register: ASYNCH_TX_DIONUM

Address: 5410

```
property asynch_tx_go: int
```

Write a 1 to this register to initiate a transmission.

Register: ASYNCH_TX_GO

Address: 5450

```
property bootloader_version: float
```

The bootloader version installed on the main processor.

Register: BOOTLOADER_VERSION

Address: 60006

```
property cio0: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: CIO0 (CIO#(0:3)@0)

Address: 2016

```
property cio1: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: CIO1 (CIO#(0:3)@1)

Address: 2017

```
property cio2: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: CIO2 (CIO#(0:3)@2)

Address: 2018

```
property cio3: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: CIO3 (CIO#(0:3)@3)

Address: 2019

```
property cio_direction: int
```

Read or write the direction of the 4 bits of CIO in a single binary-encoded value. 0=Input and 1=Output. The upper 8-bits of this value are inhibits.

Register: CIO_DIRECTION

Address: 2602

```
property cio_mio_state: int
```

Read or write the state of the 12 bits of CIO-MIO in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. Reading lines set to output returns the current logic levels on the terminals, not necessarily the output states written.

Register: CIO_MIO_STATE

Address: 2582

```
property cio_state: int
```

Read or write the state of the 4 bits of CIO in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. Reading lines set to output returns the current logic levels on the terminals, not necessarily the output states written. The upper 8-bits of this value are inhibits.

Register: CIO_STATE

Address: 2502

```
property cleanse: int
```

Writing 0x5317052E to this register will trigger a system cleanse.

Register: CLEANSE

Address: 49090

```
property core_timer: int
```

Internal 32-bit system timer running at 1/2 core speed, thus normally 80M/2 => 40 MHz.

Register: CORE_TIMER

Address: 61520

```
property current_source_10ua_cal_value: float
```

Fixed current source value in Amps for the 10UA terminal. This value is stored during factory calibration, it is not a current reading. Using the equation V=IR, with a known current and voltage, it is possible to calculate resistance of RTDs.

Register: CURRENT_SOURCE_10UA_CAL_VALUE

Address: 1900

```
property current_source_200ua_cal_value: float
```

Fixed current source value in Amps for the 200UA terminal. This value is stored during factory calibration, it is not a current reading. Using the equation V=IR, with a known current and voltage, it is possible to calculate resistance of RTDs.

Register: CURRENT_SOURCE_200UA_CAL_VALUE

Address: 1902

```
property dac0: float
```

Pass a voltage for the specified analog output.

Register: DAC0 (DAC#(0:1)@0)

Address: 1000

```
property dac0_binary: int
```

Writes binary values to the DACs. Binary values are 16-bit. If the DAC's resolution is less than 16 then the lower bits are ignored. 0 = lowest output, 65535 = highest output.

Register: DAC0_BINARY (DAC#(0:1)_BINARY@0)

Address: 51000

```
property dac1: float
```

Pass a voltage for the specified analog output.

Register: DAC1 (DAC#(0:1)@1)

Address: 1002

```
property dac1_binary: int
```

Writes binary values to the DACs. Binary values are 16-bit. If the DAC's resolution is less than 16 then the lower bits are ignored. 0 = lowest output, 65535 = highest output.

Register: DAC1_BINARY (DAC#(0:1)_BINARY@1)

Address: 51002

```
property dac1_frequency_out_enable: int
```

0 = off, 1 = output 10 Hz signal on DAC1. The signal will be a square wave with peaks of 0 and 3.3V. Note that writing to DAC1 or enabling a stream out which is targeting DAC1 will disable this feature.

Register: DAC1_FREQUENCY_OUT_ENABLE

Address: 61532

```
property device_name_default: str
```

Reads return the current device name. Writes update the default and current device name. A reboot is necessary to update the name reported by NBNS. Up to 49 characters, cannot contain periods.

Register: DEVICE_NAME_DEFAULT

Address: 60500

```
property device_reset_dbg_reg: int
```

Bitmask register that indicates why a device was last re-set. bit0: Power-on reset, bit1: Brown-out, bit2: Wake from idle, bit3: Wake from sleep, bit4: watchdog timer time-out, bit5: reserved, bit6: Software reset, bit7: external reset (MCLR), bit8: Voltage regulator standby, bit9: Configuration mismatch.

Register: DEVICE_RESET_DBG_REG

Address: 60014

```
property dio0_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO0_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@0)

Address: 44300

```
property dio0_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO0_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@0)

Address: 44400

```
property dio0_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO0_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@0)

Address: 44500

```
property dio0_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO0_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@0)

Address: 44600

```
property dio0_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO0_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@0)

Address: 45000

```
property dio0_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO0_EF_ENABLE (DIO#(0:22)_EF_ENABLE@0)

Address: 44000

```
property dio0_ef_index: int
```

An index to specify the feature you want.

Register: DIO0_EF_INDEX (DIO#(0:22)_EF_INDEX@0)

Address: 44100

```
property dio0_ef_options: int
```

Function dependent on selected feature index.

Register: DIO0_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@0)

Address: 44200

```
property dio0_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO0_EF_READ_A (DIO#(0:22)_EF_READ_A@0)

Address: 3000

```
property dio0_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO0_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@0)

Address: 3100

Type:
Reads the same value as DIO#(0

```
property dio0_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO0_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@0)

Address: 3500

```
property dio0_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO0_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@0)

Address: 3600

```
property dio0_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO0_EF_READ_B (DIO#(0:22)_EF_READ_B@0)

Address: 3200

```
property dio0_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO0_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@0)

Address: 3700

```
property dio10_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO10_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@10)

Address: 44320

```
property dio10_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO10_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@10)

Address: 44420

```
property dio10_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO10_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@10)

Address: 44520

```
property dio10_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO10_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@10)

Address: 44620

```
property dio10_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO10_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@10)

Address: 45020

```
property dio10_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO10_EF_ENABLE (DIO#(0:22)_EF_ENABLE@10)

Address: 44020

```
property dio10_ef_index: int
```

An index to specify the feature you want.

Register: DIO10_EF_INDEX (DIO#(0:22)_EF_INDEX@10)

Address: 44120

```
property dio10_ef_options: int
```

Function dependent on selected feature index.

Register: DIO10_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@10)

Address: 44220

```
property dio10_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO10_EF_READ_A (DIO#(0:22)_EF_READ_A@10)

Address: 3020

```
property dio10_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO10_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@10)

Address: 3120

Type:
Reads the same value as DIO#(0

```
property dio10_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO10_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@10)

Address: 3520

```
property dio10_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO10_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@10)

Address: 3620

```
property dio10_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO10_EF_READ_B (DIO#(0:22)_EF_READ_B@10)

Address: 3220

```
property dio10_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO10_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@10)

Address: 3720

```
property dio11_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO11_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@11)

Address: 44322

```
property dio11_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO11_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@11)

Address: 44422

```
property dio11_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO11_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@11)

Address: 44522

```
property dio11_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO11_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@11)

Address: 44622

```
property dio11_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO11_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@11)

Address: 45022

```
property dio11_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO11_EF_ENABLE (DIO#(0:22)_EF_ENABLE@11)

Address: 44022

```
property dio11_ef_index: int
```

An index to specify the feature you want.

Register: DIO11_EF_INDEX (DIO#(0:22)_EF_INDEX@11)

Address: 44122

```
property dio11_ef_options: int
```

Function dependent on selected feature index.

Register: DIO11_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@11)

Address: 44222

```
property dio11_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO11_EF_READ_A (DIO#(0:22)_EF_READ_A@11)

Address: 3022

```
property dio11_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO11_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@11)

Address: 3122

Type:
Reads the same value as DIO#(0

```
property dio11_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO11_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@11)

Address: 3522

```
property dio11_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO11_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@11)

Address: 3622

```
property dio11_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO11_EF_READ_B (DIO#(0:22)_EF_READ_B@11)

Address: 3222

```
property dio11_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO11_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@11)

Address: 3722

```
property dio12_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO12_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@12)

Address: 44324

```
property dio12_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO12_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@12)

Address: 44424

```
property dio12_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO12_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@12)

Address: 44524

```
property dio12_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO12_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@12)

Address: 44624

```
property dio12_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO12_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@12)

Address: 45024

```
property dio12_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO12_EF_ENABLE (DIO#(0:22)_EF_ENABLE@12)

Address: 44024

```
property dio12_ef_index: int
```

An index to specify the feature you want.

Register: DIO12_EF_INDEX (DIO#(0:22)_EF_INDEX@12)

Address: 44124

```
property dio12_ef_options: int
```

Function dependent on selected feature index.

Register: DIO12_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@12)

Address: 44224

```
property dio12_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO12_EF_READ_A (DIO#(0:22)_EF_READ_A@12)

Address: 3024

```
property dio12_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO12_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@12)

Address: 3124

Type:
Reads the same value as DIO#(0

```
property dio12_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO12_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@12)

Address: 3524

```
property dio12_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO12_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@12)

Address: 3624

```
property dio12_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO12_EF_READ_B (DIO#(0:22)_EF_READ_B@12)

Address: 3224

```
property dio12_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO12_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@12)

Address: 3724

```
property dio13_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO13_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@13)

Address: 44326

```
property dio13_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO13_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@13)

Address: 44426

```
property dio13_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO13_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@13)

Address: 44526

```
property dio13_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO13_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@13)

Address: 44626

```
property dio13_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO13_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@13)

Address: 45026

```
property dio13_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO13_EF_ENABLE (DIO#(0:22)_EF_ENABLE@13)

Address: 44026

```
property dio13_ef_index: int
```

An index to specify the feature you want.

Register: DIO13_EF_INDEX (DIO#(0:22)_EF_INDEX@13)

Address: 44126

```
property dio13_ef_options: int
```

Function dependent on selected feature index.

Register: DIO13_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@13)

Address: 44226

```
property dio13_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO13_EF_READ_A (DIO#(0:22)_EF_READ_A@13)

Address: 3026

```
property dio13_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO13_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@13)

Address: 3126

Type:
Reads the same value as DIO#(0

```
property dio13_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO13_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@13)

Address: 3526

```
property dio13_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO13_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@13)

Address: 3626

```
property dio13_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO13_EF_READ_B (DIO#(0:22)_EF_READ_B@13)

Address: 3226

```
property dio13_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO13_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@13)

Address: 3726

```
property dio14_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO14_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@14)

Address: 44328

```
property dio14_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO14_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@14)

Address: 44428

```
property dio14_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO14_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@14)

Address: 44528

```
property dio14_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO14_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@14)

Address: 44628

```
property dio14_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO14_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@14)

Address: 45028

```
property dio14_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO14_EF_ENABLE (DIO#(0:22)_EF_ENABLE@14)

Address: 44028

```
property dio14_ef_index: int
```

An index to specify the feature you want.

Register: DIO14_EF_INDEX (DIO#(0:22)_EF_INDEX@14)

Address: 44128

```
property dio14_ef_options: int
```

Function dependent on selected feature index.

Register: DIO14_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@14)

Address: 44228

```
property dio14_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO14_EF_READ_A (DIO#(0:22)_EF_READ_A@14)

Address: 3028

```
property dio14_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO14_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@14)

Address: 3128

Type:
Reads the same value as DIO#(0

```
property dio14_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO14_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@14)

Address: 3528

```
property dio14_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO14_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@14)

Address: 3628

```
property dio14_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO14_EF_READ_B (DIO#(0:22)_EF_READ_B@14)

Address: 3228

```
property dio14_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO14_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@14)

Address: 3728

```
property dio15_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO15_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@15)

Address: 44330

```
property dio15_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO15_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@15)

Address: 44430

```
property dio15_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO15_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@15)

Address: 44530

```
property dio15_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO15_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@15)

Address: 44630

```
property dio15_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO15_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@15)

Address: 45030

```
property dio15_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO15_EF_ENABLE (DIO#(0:22)_EF_ENABLE@15)

Address: 44030

```
property dio15_ef_index: int
```

An index to specify the feature you want.

Register: DIO15_EF_INDEX (DIO#(0:22)_EF_INDEX@15)

Address: 44130

```
property dio15_ef_options: int
```

Function dependent on selected feature index.

Register: DIO15_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@15)

Address: 44230

```
property dio15_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO15_EF_READ_A (DIO#(0:22)_EF_READ_A@15)

Address: 3030

```
property dio15_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO15_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@15)

Address: 3130

Type:
Reads the same value as DIO#(0

```
property dio15_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO15_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@15)

Address: 3530

```
property dio15_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO15_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@15)

Address: 3630

```
property dio15_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO15_EF_READ_B (DIO#(0:22)_EF_READ_B@15)

Address: 3230

```
property dio15_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO15_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@15)

Address: 3730

```
property dio16_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO16_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@16)

Address: 44332

```
property dio16_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO16_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@16)

Address: 44432

```
property dio16_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO16_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@16)

Address: 44532

```
property dio16_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO16_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@16)

Address: 44632

```
property dio16_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO16_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@16)

Address: 45032

```
property dio16_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO16_EF_ENABLE (DIO#(0:22)_EF_ENABLE@16)

Address: 44032

```
property dio16_ef_index: int
```

An index to specify the feature you want.

Register: DIO16_EF_INDEX (DIO#(0:22)_EF_INDEX@16)

Address: 44132

```
property dio16_ef_options: int
```

Function dependent on selected feature index.

Register: DIO16_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@16)

Address: 44232

```
property dio16_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO16_EF_READ_A (DIO#(0:22)_EF_READ_A@16)

Address: 3032

```
property dio16_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO16_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@16)

Address: 3132

Type:
Reads the same value as DIO#(0

```
property dio16_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO16_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@16)

Address: 3532

```
property dio16_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO16_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@16)

Address: 3632

```
property dio16_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO16_EF_READ_B (DIO#(0:22)_EF_READ_B@16)

Address: 3232

```
property dio16_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO16_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@16)

Address: 3732

```
property dio17_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO17_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@17)

Address: 44334

```
property dio17_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO17_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@17)

Address: 44434

```
property dio17_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO17_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@17)

Address: 44534

```
property dio17_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO17_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@17)

Address: 44634

```
property dio17_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO17_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@17)

Address: 45034

```
property dio17_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO17_EF_ENABLE (DIO#(0:22)_EF_ENABLE@17)

Address: 44034

```
property dio17_ef_index: int
```

An index to specify the feature you want.

Register: DIO17_EF_INDEX (DIO#(0:22)_EF_INDEX@17)

Address: 44134

```
property dio17_ef_options: int
```

Function dependent on selected feature index.

Register: DIO17_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@17)

Address: 44234

```
property dio17_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO17_EF_READ_A (DIO#(0:22)_EF_READ_A@17)

Address: 3034

```
property dio17_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO17_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@17)

Address: 3134

Type:
Reads the same value as DIO#(0

```
property dio17_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO17_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@17)

Address: 3534

```
property dio17_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO17_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@17)

Address: 3634

```
property dio17_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO17_EF_READ_B (DIO#(0:22)_EF_READ_B@17)

Address: 3234

```
property dio17_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO17_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@17)

Address: 3734

```
property dio18_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO18_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@18)

Address: 44336

```
property dio18_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO18_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@18)

Address: 44436

```
property dio18_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO18_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@18)

Address: 44536

```
property dio18_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO18_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@18)

Address: 44636

```
property dio18_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO18_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@18)

Address: 45036

```
property dio18_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO18_EF_ENABLE (DIO#(0:22)_EF_ENABLE@18)

Address: 44036

```
property dio18_ef_index: int
```

An index to specify the feature you want.

Register: DIO18_EF_INDEX (DIO#(0:22)_EF_INDEX@18)

Address: 44136

```
property dio18_ef_options: int
```

Function dependent on selected feature index.

Register: DIO18_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@18)

Address: 44236

```
property dio18_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO18_EF_READ_A (DIO#(0:22)_EF_READ_A@18)

Address: 3036

```
property dio18_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO18_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@18)

Address: 3136

Type:
Reads the same value as DIO#(0

```
property dio18_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO18_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@18)

Address: 3536

```
property dio18_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO18_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@18)

Address: 3636

```
property dio18_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO18_EF_READ_B (DIO#(0:22)_EF_READ_B@18)

Address: 3236

```
property dio18_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO18_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@18)

Address: 3736

```
property dio19_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO19_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@19)

Address: 44338

```
property dio19_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO19_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@19)

Address: 44438

```
property dio19_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO19_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@19)

Address: 44538

```
property dio19_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO19_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@19)

Address: 44638

```
property dio19_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO19_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@19)

Address: 45038

```
property dio19_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO19_EF_ENABLE (DIO#(0:22)_EF_ENABLE@19)

Address: 44038

```
property dio19_ef_index: int
```

An index to specify the feature you want.

Register: DIO19_EF_INDEX (DIO#(0:22)_EF_INDEX@19)

Address: 44138

```
property dio19_ef_options: int
```

Function dependent on selected feature index.

Register: DIO19_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@19)

Address: 44238

```
property dio19_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO19_EF_READ_A (DIO#(0:22)_EF_READ_A@19)

Address: 3038

```
property dio19_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO19_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@19)

Address: 3138

Type:
Reads the same value as DIO#(0

```
property dio19_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO19_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@19)

Address: 3538

```
property dio19_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO19_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@19)

Address: 3638

```
property dio19_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO19_EF_READ_B (DIO#(0:22)_EF_READ_B@19)

Address: 3238

```
property dio19_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO19_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@19)

Address: 3738

```
property dio1_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO1_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@1)

Address: 44302

```
property dio1_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO1_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@1)

Address: 44402

```
property dio1_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO1_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@1)

Address: 44502

```
property dio1_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO1_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@1)

Address: 44602

```
property dio1_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO1_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@1)

Address: 45002

```
property dio1_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO1_EF_ENABLE (DIO#(0:22)_EF_ENABLE@1)

Address: 44002

```
property dio1_ef_index: int
```

An index to specify the feature you want.

Register: DIO1_EF_INDEX (DIO#(0:22)_EF_INDEX@1)

Address: 44102

```
property dio1_ef_options: int
```

Function dependent on selected feature index.

Register: DIO1_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@1)

Address: 44202

```
property dio1_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO1_EF_READ_A (DIO#(0:22)_EF_READ_A@1)

Address: 3002

```
property dio1_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO1_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@1)

Address: 3102

Type:
Reads the same value as DIO#(0

```
property dio1_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO1_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@1)

Address: 3502

```
property dio1_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO1_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@1)

Address: 3602

```
property dio1_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO1_EF_READ_B (DIO#(0:22)_EF_READ_B@1)

Address: 3202

```
property dio1_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO1_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@1)

Address: 3702

```
property dio20_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO20_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@20)

Address: 44340

```
property dio20_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO20_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@20)

Address: 44440

```
property dio20_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO20_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@20)

Address: 44540

```
property dio20_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO20_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@20)

Address: 44640

```
property dio20_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO20_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@20)

Address: 45040

```
property dio20_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO20_EF_ENABLE (DIO#(0:22)_EF_ENABLE@20)

Address: 44040

```
property dio20_ef_index: int
```

An index to specify the feature you want.

Register: DIO20_EF_INDEX (DIO#(0:22)_EF_INDEX@20)

Address: 44140

```
property dio20_ef_options: int
```

Function dependent on selected feature index.

Register: DIO20_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@20)

Address: 44240

```
property dio20_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO20_EF_READ_A (DIO#(0:22)_EF_READ_A@20)

Address: 3040

```
property dio20_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO20_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@20)

Address: 3140

Type:
Reads the same value as DIO#(0

```
property dio20_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO20_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@20)

Address: 3540

```
property dio20_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO20_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@20)

Address: 3640

```
property dio20_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO20_EF_READ_B (DIO#(0:22)_EF_READ_B@20)

Address: 3240

```
property dio20_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO20_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@20)

Address: 3740

```
property dio21_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO21_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@21)

Address: 44342

```
property dio21_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO21_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@21)

Address: 44442

```
property dio21_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO21_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@21)

Address: 44542

```
property dio21_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO21_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@21)

Address: 44642

```
property dio21_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO21_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@21)

Address: 45042

```
property dio21_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO21_EF_ENABLE (DIO#(0:22)_EF_ENABLE@21)

Address: 44042

```
property dio21_ef_index: int
```

An index to specify the feature you want.

Register: DIO21_EF_INDEX (DIO#(0:22)_EF_INDEX@21)

Address: 44142

```
property dio21_ef_options: int
```

Function dependent on selected feature index.

Register: DIO21_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@21)

Address: 44242

```
property dio21_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO21_EF_READ_A (DIO#(0:22)_EF_READ_A@21)

Address: 3042

```
property dio21_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO21_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@21)

Address: 3142

Type:
Reads the same value as DIO#(0

```
property dio21_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO21_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@21)

Address: 3542

```
property dio21_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO21_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@21)

Address: 3642

```
property dio21_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO21_EF_READ_B (DIO#(0:22)_EF_READ_B@21)

Address: 3242

```
property dio21_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO21_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@21)

Address: 3742

```
property dio22_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO22_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@22)

Address: 44344

```
property dio22_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO22_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@22)

Address: 44444

```
property dio22_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO22_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@22)

Address: 44544

```
property dio22_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO22_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@22)

Address: 44644

```
property dio22_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO22_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@22)

Address: 45044

```
property dio22_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO22_EF_ENABLE (DIO#(0:22)_EF_ENABLE@22)

Address: 44044

```
property dio22_ef_index: int
```

An index to specify the feature you want.

Register: DIO22_EF_INDEX (DIO#(0:22)_EF_INDEX@22)

Address: 44144

```
property dio22_ef_options: int
```

Function dependent on selected feature index.

Register: DIO22_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@22)

Address: 44244

```
property dio22_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO22_EF_READ_A (DIO#(0:22)_EF_READ_A@22)

Address: 3044

```
property dio22_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO22_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@22)

Address: 3144

Type:
Reads the same value as DIO#(0

```
property dio22_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO22_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@22)

Address: 3544

```
property dio22_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO22_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@22)

Address: 3644

```
property dio22_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO22_EF_READ_B (DIO#(0:22)_EF_READ_B@22)

Address: 3244

```
property dio22_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO22_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@22)

Address: 3744

```
property dio2_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO2_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@2)

Address: 44304

```
property dio2_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO2_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@2)

Address: 44404

```
property dio2_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO2_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@2)

Address: 44504

```
property dio2_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO2_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@2)

Address: 44604

```
property dio2_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO2_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@2)

Address: 45004

```
property dio2_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO2_EF_ENABLE (DIO#(0:22)_EF_ENABLE@2)

Address: 44004

```
property dio2_ef_index: int
```

An index to specify the feature you want.

Register: DIO2_EF_INDEX (DIO#(0:22)_EF_INDEX@2)

Address: 44104

```
property dio2_ef_options: int
```

Function dependent on selected feature index.

Register: DIO2_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@2)

Address: 44204

```
property dio2_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO2_EF_READ_A (DIO#(0:22)_EF_READ_A@2)

Address: 3004

```
property dio2_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO2_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@2)

Address: 3104

Type:
Reads the same value as DIO#(0

```
property dio2_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO2_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@2)

Address: 3504

```
property dio2_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO2_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@2)

Address: 3604

```
property dio2_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO2_EF_READ_B (DIO#(0:22)_EF_READ_B@2)

Address: 3204

```
property dio2_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO2_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@2)

Address: 3704

```
property dio3_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO3_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@3)

Address: 44306

```
property dio3_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO3_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@3)

Address: 44406

```
property dio3_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO3_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@3)

Address: 44506

```
property dio3_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO3_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@3)

Address: 44606

```
property dio3_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO3_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@3)

Address: 45006

```
property dio3_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO3_EF_ENABLE (DIO#(0:22)_EF_ENABLE@3)

Address: 44006

```
property dio3_ef_index: int
```

An index to specify the feature you want.

Register: DIO3_EF_INDEX (DIO#(0:22)_EF_INDEX@3)

Address: 44106

```
property dio3_ef_options: int
```

Function dependent on selected feature index.

Register: DIO3_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@3)

Address: 44206

```
property dio3_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO3_EF_READ_A (DIO#(0:22)_EF_READ_A@3)

Address: 3006

```
property dio3_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO3_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@3)

Address: 3106

Type:
Reads the same value as DIO#(0

```
property dio3_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO3_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@3)

Address: 3506

```
property dio3_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO3_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@3)

Address: 3606

```
property dio3_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO3_EF_READ_B (DIO#(0:22)_EF_READ_B@3)

Address: 3206

```
property dio3_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO3_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@3)

Address: 3706

```
property dio4_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO4_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@4)

Address: 44308

```
property dio4_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO4_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@4)

Address: 44408

```
property dio4_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO4_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@4)

Address: 44508

```
property dio4_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO4_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@4)

Address: 44608

```
property dio4_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO4_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@4)

Address: 45008

```
property dio4_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO4_EF_ENABLE (DIO#(0:22)_EF_ENABLE@4)

Address: 44008

```
property dio4_ef_index: int
```

An index to specify the feature you want.

Register: DIO4_EF_INDEX (DIO#(0:22)_EF_INDEX@4)

Address: 44108

```
property dio4_ef_options: int
```

Function dependent on selected feature index.

Register: DIO4_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@4)

Address: 44208

```
property dio4_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO4_EF_READ_A (DIO#(0:22)_EF_READ_A@4)

Address: 3008

```
property dio4_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO4_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@4)

Address: 3108

Type:
Reads the same value as DIO#(0

```
property dio4_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO4_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@4)

Address: 3508

```
property dio4_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO4_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@4)

Address: 3608

```
property dio4_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO4_EF_READ_B (DIO#(0:22)_EF_READ_B@4)

Address: 3208

```
property dio4_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO4_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@4)

Address: 3708

```
property dio5_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO5_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@5)

Address: 44310

```
property dio5_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO5_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@5)

Address: 44410

```
property dio5_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO5_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@5)

Address: 44510

```
property dio5_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO5_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@5)

Address: 44610

```
property dio5_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO5_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@5)

Address: 45010

```
property dio5_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO5_EF_ENABLE (DIO#(0:22)_EF_ENABLE@5)

Address: 44010

```
property dio5_ef_index: int
```

An index to specify the feature you want.

Register: DIO5_EF_INDEX (DIO#(0:22)_EF_INDEX@5)

Address: 44110

```
property dio5_ef_options: int
```

Function dependent on selected feature index.

Register: DIO5_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@5)

Address: 44210

```
property dio5_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO5_EF_READ_A (DIO#(0:22)_EF_READ_A@5)

Address: 3010

```
property dio5_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO5_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@5)

Address: 3110

Type:
Reads the same value as DIO#(0

```
property dio5_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO5_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@5)

Address: 3510

```
property dio5_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO5_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@5)

Address: 3610

```
property dio5_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO5_EF_READ_B (DIO#(0:22)_EF_READ_B@5)

Address: 3210

```
property dio5_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO5_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@5)

Address: 3710

```
property dio6_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO6_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@6)

Address: 44312

```
property dio6_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO6_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@6)

Address: 44412

```
property dio6_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO6_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@6)

Address: 44512

```
property dio6_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO6_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@6)

Address: 44612

```
property dio6_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO6_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@6)

Address: 45012

```
property dio6_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO6_EF_ENABLE (DIO#(0:22)_EF_ENABLE@6)

Address: 44012

```
property dio6_ef_index: int
```

An index to specify the feature you want.

Register: DIO6_EF_INDEX (DIO#(0:22)_EF_INDEX@6)

Address: 44112

```
property dio6_ef_options: int
```

Function dependent on selected feature index.

Register: DIO6_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@6)

Address: 44212

```
property dio6_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO6_EF_READ_A (DIO#(0:22)_EF_READ_A@6)

Address: 3012

```
property dio6_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO6_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@6)

Address: 3112

Type:
Reads the same value as DIO#(0

```
property dio6_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO6_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@6)

Address: 3512

```
property dio6_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO6_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@6)

Address: 3612

```
property dio6_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO6_EF_READ_B (DIO#(0:22)_EF_READ_B@6)

Address: 3212

```
property dio6_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO6_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@6)

Address: 3712

```
property dio7_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO7_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@7)

Address: 44314

```
property dio7_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO7_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@7)

Address: 44414

```
property dio7_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO7_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@7)

Address: 44514

```
property dio7_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO7_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@7)

Address: 44614

```
property dio7_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO7_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@7)

Address: 45014

```
property dio7_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO7_EF_ENABLE (DIO#(0:22)_EF_ENABLE@7)

Address: 44014

```
property dio7_ef_index: int
```

An index to specify the feature you want.

Register: DIO7_EF_INDEX (DIO#(0:22)_EF_INDEX@7)

Address: 44114

```
property dio7_ef_options: int
```

Function dependent on selected feature index.

Register: DIO7_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@7)

Address: 44214

```
property dio7_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO7_EF_READ_A (DIO#(0:22)_EF_READ_A@7)

Address: 3014

```
property dio7_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO7_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@7)

Address: 3114

Type:
Reads the same value as DIO#(0

```
property dio7_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO7_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@7)

Address: 3514

```
property dio7_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO7_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@7)

Address: 3614

```
property dio7_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO7_EF_READ_B (DIO#(0:22)_EF_READ_B@7)

Address: 3214

```
property dio7_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO7_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@7)

Address: 3714

```
property dio8_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO8_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@8)

Address: 44316

```
property dio8_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO8_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@8)

Address: 44416

```
property dio8_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO8_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@8)

Address: 44516

```
property dio8_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO8_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@8)

Address: 44616

```
property dio8_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO8_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@8)

Address: 45016

```
property dio8_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO8_EF_ENABLE (DIO#(0:22)_EF_ENABLE@8)

Address: 44016

```
property dio8_ef_index: int
```

An index to specify the feature you want.

Register: DIO8_EF_INDEX (DIO#(0:22)_EF_INDEX@8)

Address: 44116

```
property dio8_ef_options: int
```

Function dependent on selected feature index.

Register: DIO8_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@8)

Address: 44216

```
property dio8_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO8_EF_READ_A (DIO#(0:22)_EF_READ_A@8)

Address: 3016

```
property dio8_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO8_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@8)

Address: 3116

Type:
Reads the same value as DIO#(0

```
property dio8_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO8_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@8)

Address: 3516

```
property dio8_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO8_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@8)

Address: 3616

```
property dio8_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO8_EF_READ_B (DIO#(0:22)_EF_READ_B@8)

Address: 3216

```
property dio8_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO8_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@8)

Address: 3716

```
property dio9_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO9_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@9)

Address: 44318

```
property dio9_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO9_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@9)

Address: 44418

```
property dio9_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO9_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@9)

Address: 44518

```
property dio9_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO9_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@9)

Address: 44618

```
property dio9_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO9_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@9)

Address: 45018

```
property dio9_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO9_EF_ENABLE (DIO#(0:22)_EF_ENABLE@9)

Address: 44018

```
property dio9_ef_index: int
```

An index to specify the feature you want.

Register: DIO9_EF_INDEX (DIO#(0:22)_EF_INDEX@9)

Address: 44118

```
property dio9_ef_options: int
```

Function dependent on selected feature index.

Register: DIO9_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@9)

Address: 44218

```
property dio9_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO9_EF_READ_A (DIO#(0:22)_EF_READ_A@9)

Address: 3018

```
property dio9_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO9_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@9)

Address: 3118

Type:
Reads the same value as DIO#(0

```
property dio9_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO9_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@9)

Address: 3518

```
property dio9_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO9_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@9)

Address: 3618

```
property dio9_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO9_EF_READ_B (DIO#(0:22)_EF_READ_B@9)

Address: 3218

```
property dio9_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO9_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@9)

Address: 3718

```
property dio_direction: int
```

Read or write the direction of all digital I/O in a single binary-encoded value. 0=Input and 1=Output. Writes are filtered by the value in DIO_INHIBIT.

Register: DIO_DIRECTION

Address: 2850

```
property dio_ef_clock0_count: int
```

Current tick count of this clock. Will read between 0 and ROLL_VALUE-1.

Register: DIO_EF_CLOCK0_COUNT

Address: 44908

```
property dio_ef_clock0_divisor: int
```

Divides the core clock. Valid options: 1,2,4,8,16,32,64,256.

Register: DIO_EF_CLOCK0_DIVISOR

Address: 44901

```
property dio_ef_clock0_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration.

Register: DIO_EF_CLOCK0_ENABLE

Address: 44900

```
property dio_ef_clock0_options: int
```

Bitmask: bit0: 1 = use external clock. All other bits reserved.

Register: DIO_EF_CLOCK0_OPTIONS

Address: 44902

```
property dio_ef_clock0_roll_value: int
```

The clock will count to this value and then start over at zero. The clock pulses counted are those after the divisor. 0 results in the max roll value possible. This is a 32-bit value (0-4294967295) if using a 32-bit clock, and a 16-bit value (0-65535) if using a 16-bit clock.

Register: DIO_EF_CLOCK0_ROLL_VALUE

Address: 44904

```
property dio_ef_clock1_count: int
```

Current tick count of this clock. Will read between 0 and ROLL_VALUE-1.

Register: DIO_EF_CLOCK1_COUNT

Address: 44918

```
property dio_ef_clock1_divisor: int
```

Divides the core clock. Valid options: 1,2,4,8,16,32,64,256.

Register: DIO_EF_CLOCK1_DIVISOR

Address: 44911

```
property dio_ef_clock1_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration.

Register: DIO_EF_CLOCK1_ENABLE

Address: 44910

```
property dio_ef_clock1_options: int
```

Bitmask: bit0: 1 = use external clock. All other bits reserved.

Register: DIO_EF_CLOCK1_OPTIONS

Address: 44912

```
property dio_ef_clock1_roll_value: int
```

The clock will count to this value and then start over at zero. The clock pulses counted are those after the divisor. 0 results in the max roll value possible. This is a 32-bit value (0-4294967295) if using a 32-bit clock, and a 16-bit value (0-65535) if using a 16-bit clock.

Register: DIO_EF_CLOCK1_ROLL_VALUE

Address: 44914

```
property dio_ef_clock2_count: int
```

Current tick count of this clock. Will read between 0 and ROLL_VALUE-1.

Register: DIO_EF_CLOCK2_COUNT

Address: 44928

```
property dio_ef_clock2_divisor: int
```

Divides the core clock. Valid options: 1,2,4,8,16,32,64,256.

Register: DIO_EF_CLOCK2_DIVISOR

Address: 44921

```
property dio_ef_clock2_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration.

Register: DIO_EF_CLOCK2_ENABLE

Address: 44920

```
property dio_ef_clock2_options: int
```

Bitmask: bit0: 1 = use external clock. All other bits reserved.

Register: DIO_EF_CLOCK2_OPTIONS

Address: 44922

```
property dio_ef_clock2_roll_value: int
```

The clock will count to this value and then start over at zero. The clock pulses counted are those after the divisor. 0 results in the max roll value possible. This is a 32-bit value (0-4294967295) if using a 32-bit clock, and a 16-bit value (0-65535) if using a 16-bit clock.

Register: DIO_EF_CLOCK2_ROLL_VALUE

Address: 44924

```
property dio_inhibit: int
```

A single binary-encoded value where each bit determines whether _STATE, _DIRECTION or _ANALOG_ENABLE writes affect that bit of digital I/O. 0=Default=Affected, 1=Ignored.

Register: DIO_INHIBIT

Address: 2900

```
property dio_state: int
```

Read or write the state of all digital I/O in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. A read of an output returns the current logic level on the terminal, not necessarily the output state written. Writes are filtered by the value in DIO_INHIBIT.

Register: DIO_STATE

Address: 2800

```
property eio0: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: EIO0 (EIO#(0:7)@0)

Address: 2008

```
property eio1: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: EIO1 (EIO#(0:7)@1)

Address: 2009

```
property eio2: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: EIO2 (EIO#(0:7)@2)

Address: 2010

```
property eio3: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: EIO3 (EIO#(0:7)@3)

Address: 2011

```
property eio4: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: EIO4 (EIO#(0:7)@4)

Address: 2012

```
property eio5: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: EIO5 (EIO#(0:7)@5)

Address: 2013

```
property eio6: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: EIO6 (EIO#(0:7)@6)

Address: 2014

```
property eio7: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: EIO7 (EIO#(0:7)@7)

Address: 2015

```
property eio_cio_state: int
```

Read or write the state of the 12 bits of EIO-CIO in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. Reading lines set to output returns the current logic levels on the terminals, not necessarily the output states written. As of firmware 1.0172, MIO states are included in the upper nibble of the CIO byte.

Register: EIO_CIO_STATE

Address: 2581

```
property eio_direction: int
```

Read or write the direction of the 8 bits of EIO in a single binary-encoded value. 0=Input and 1=Output. The upper 8-bits of this value are inhibits.

Register: EIO_DIRECTION

Address: 2601

```
property eio_state: int
```

Read or write the state of the 8 bits of EIO in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. Reading lines set to output returns the current logic levels on the terminals, not necessarily the output states written. The upper 8-bits of this value are inhibits.

Register: EIO_STATE

Address: 2501

```
property ethernet_altdns: str
```

Read the current Alt DNS of wired Ethernet.

Register: ETHERNET_ALTDNS

Address: 49108

```
property ethernet_altdns_default: str
```

The Alt DNS of wired Ethernet after a power-cycle to the device.

Register: ETHERNET_ALTDNS_DEFAULT

Address: 49158

```
property ethernet_apply_settings: int
```

Writing 1 to this register power-cycles Ethernet. It tells the device to waits 1s before turning off Ethernet and then 500ms before turning it back on.

Register: ETHERNET_APPLY_SETTINGS

Address: 49190

```
property ethernet_dhcp_enable: int
```

Read the current Enabled/Disabled state of Ethernet DHCP.

Register: ETHERNET_DHCP_ENABLE

Address: 49110

```
property ethernet_dhcp_enable_default: int
```

The Enabled/Disabled state of Ethernet DHCP after a power-cycle to the device.

Register: ETHERNET_DHCP_ENABLE_DEFAULT

Address: 49160

```
property ethernet_dns: str
```

Read the current DNS of wired Ethernet.

Register: ETHERNET_DNS

Address: 49106

```
property ethernet_dns_default: str
```

The DNS of wired Ethernet after a power-cycle to the device.

Register: ETHERNET_DNS_DEFAULT

Address: 49156

```
property ethernet_gateway: str
```

Read the current gateway of wired Ethernet.

Register: ETHERNET_GATEWAY

Address: 49104

```
property ethernet_gateway_default: str
```

The gateway of wired Ethernet after a power-cycle to the device.

Register: ETHERNET_GATEWAY_DEFAULT

Address: 49154

```
property ethernet_ip: str
```

Read the current IP address of wired Ethernet.

Register: ETHERNET_IP

Address: 49100

```
property ethernet_ip_default: str
```

The IP address of wired Ethernet after a power-cycle to the device.

Register: ETHERNET_IP_DEFAULT

Address: 49150

```
property ethernet_mac: str
```

The MAC address of the wired Ethernet module.

Register: ETHERNET_MAC

Address: 60020

```
property ethernet_subnet: str
```

Read the current subnet of wired Ethernet.

Register: ETHERNET_SUBNET

Address: 49102

```
property ethernet_subnet_default: str
```

The subnet of wired Ethernet after a power-cycle to the device.

Register: ETHERNET_SUBNET_DEFAULT

Address: 49152

```
property ethernet_udp_discovery_only: int
```

Restricts which Modbus operations are allowed over UDP. When set to 1, only the registers needed for discovering units can be read and any other operation will throw an error.

Register: ETHERNET_UDP_DISCOVERY_ONLY

Address: 49115

```
property ethernet_udp_discovery_only_default: int
```

The Enabled/Disabled state of ETHERNET_UDP_DISCOVERY_ONLY after a power-cycle to the device.

Register: ETHERNET_UDP_DISCOVERY_ONLY_DEFAULT

Address: 49165

```
property file_io_attributes: int
```

Used to differentiate files from directories/folders. Bitmask: Bit0: Reserved, Bit1: Reserved, Bit2: Reserved, Bit3: Reserved, Bit4: 1=Directory, Bit5: 1=File.

Register: FILE_IO_ATTRIBUTES

Address: 60623

```
property file_io_close: int
```

Write any value to this register to close the open file.

Register: FILE_IO_CLOSE

Address: 60621

```
property file_io_delete: int
```

Write any value to this register to delete the active file. Must first designate which file to delete by writing to FILE_IO_PATH_WRITE_LEN_BYTES then FILE_IO_PATH_WRITE.

Register: FILE_IO_DELETE

Address: 60622

```
property file_io_dir_change: int
```

Write any value to this register to change the current working directory (CWD). Must first designate which directory to open by writing to FILE_IO_PATH_WRITE_LEN_BYTES then FILE_IO_PATH_WRITE.

Register: FILE_IO_DIR_CHANGE

Address: 60600

```
property file_io_dir_current: int
```

Write any value to this register to load the current working directory into FILE_IO_PATH_READ, and its length into FILE_IO_PATH_READ_LEN_BYTES. Can be used to identify current position in a file tree.

Register: FILE_IO_DIR_CURRENT

Address: 60601

```
property file_io_dir_first: int
```

Write any value to this register to initiate iteration through files and directories in the CWD. Typical sequence: FILE_IO_DIR_FIRST(W), then loop through: [FILE_IO_NAME_READ_LEN(R), FILE_IO_NAME_READ(R), FILE_IO_ATTRIBUTES(R), FILE_IO_SIZE_BYTES(R), FILE_IO_DIR_NEXT(W)] ..until any of the following errors: FILE_IO_END_OF_CWD (2966), FILE_IO_INVALID_OBJECT (2809), or FILE_IO_NOT_FOUND (2960).

Register: FILE_IO_DIR_FIRST

Address: 60610

```
property file_io_dir_make: int
```

Unimplemented.

Register: FILE_IO_DIR_MAKE

Address: 60602

```
property file_io_dir_next: int
```

Write any value to this register to continue iteration through files and directories in the CWD.

Register: FILE_IO_DIR_NEXT

Address: 60611

```
property file_io_dir_remove: int
```

Unimplemented.

Register: FILE_IO_DIR_REMOVE

Address: 60603

```
property file_io_disk_format_index: int
```

Used to determine the format of the SD card. 0=None or Unknown, 1=FAT12, 2=FAT16(Windows FAT), 3=FAT32.

Register: FILE_IO_DISK_FORMAT_INDEX

Address: 60638

```
property file_io_disk_free_clusters: int
```

Free (available) clusters in the SD card. Used to determine free space. Captured on read of FILE_IO_DISK_SECTOR_SIZE_BYTES.

Register: FILE_IO_DISK_FREE_CLUSTERS

Address: 60636

```
property file_io_disk_sector_size_bytes: int
```

The size of each sector in the SD card in bytes. In Windows this is called the Allocation Size.

Register: FILE_IO_DISK_SECTOR_SIZE_BYTES

Address: 60630

```
property file_io_disk_sectors_per_cluster: int
```

The number of sectors in each cluster. Captured on read of FILE_IO_DISK_SECTOR_SIZE_BYTES.

Register: FILE_IO_DISK_SECTORS_PER_CLUSTER

Address: 60632

```
property file_io_disk_total_clusters: int
```

The total number of clusters in the SD card. Captured on read of FILE_IO_DISK_SECTOR_SIZE_BYTES.

Register: FILE_IO_DISK_TOTAL_CLUSTERS

Address: 60634

```
property file_io_open: int
```

Write any value to this register to open a file. Must first designate which file to open by writing to FILE_IO_PATH_WRITE_LEN_BYTES then FILE_IO_PATH_WRITE.

Register: FILE_IO_OPEN

Address: 60620

```
property file_io_path_read: int
```

Read the next file path in the CWD. Length of the string (in bytes) determined by FILE_IO_PATH_READ_LEN_BYTES. File paths will be null terminated.

This register is a buffer. Underrun behavior - fill with zeros.

Register: FILE_IO_PATH_READ

Address: 60652

```
property file_io_path_read_len_bytes: int
```

Read the length (in bytes) of the next file path or directory to access.

Register: FILE_IO_PATH_READ_LEN_BYTES

Address: 60642

```
property file_io_path_write: int
```

Write the desired file path. Must first write the length of the file path string (in bytes) to FILE_IO_PATH_WRITE_LEN_BYTES. File paths should be null terminated. This register is a buffer.

Register: FILE_IO_PATH_WRITE

Address: 60650

```
property file_io_path_write_len_bytes: int
```

Write the length (in bytes) of the file path or directory to access.

Register: FILE_IO_PATH_WRITE_LEN_BYTES

Address: 60640

```
property file_io_read: int
```

Read the contents of a file. Must first write to FILE_IO_OPEN. Size of the file (in bytes) determined by FILE_IO_SIZE_BYTES.

This register is a buffer. Underrun behavior - throws an error.

Register: FILE_IO_READ

Address: 60656

```
property file_io_size_bytes: int
```

The size of the file in bytes. Directories have 0 size.

Register: FILE_IO_SIZE_BYTES

Address: 60628

```
property file_io_write: int
```

Unimplemented. This register is a buffer.

Register: FILE_IO_WRITE

Address: 60654

```
property fio0: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: FIO0 (FIO#(0:7)@0)

Address: 2000

```
property fio1: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: FIO1 (FIO#(0:7)@1)

Address: 2001

```
property fio2: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: FIO2 (FIO#(0:7)@2)

Address: 2002

```
property fio3: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: FIO3 (FIO#(0:7)@3)

Address: 2003

```
property fio4: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: FIO4 (FIO#(0:7)@4)

Address: 2004

```
property fio5: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: FIO5 (FIO#(0:7)@5)

Address: 2005

```
property fio6: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: FIO6 (FIO#(0:7)@6)

Address: 2006

```
property fio7: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: FIO7 (FIO#(0:7)@7)

Address: 2007

```
property fio_direction: int
```

Read or write the direction of the 8 bits of FIO in a single binary-encoded value. 0=Input and 1=Output. The upper 8-bits of this value are inhibits.

Register: FIO_DIRECTION

Address: 2600

```
property fio_eio_state: int
```

Read or write the state of the 16 bits of FIO-EIO in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. Reading lines set to output returns the current logic levels on the terminals, not necessarily the output states written.

Register: FIO_EIO_STATE

Address: 2580

```
property fio_state: int
```

Read or write the state of the 8 bits of FIO in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. Reading lines set to output returns the current logic levels on the terminals, not necessarily the output states written. The upper 8-bits of this value are inhibits.

Register: FIO_STATE

Address: 2500

```
property firmware_version: float
```

The current firmware version installed on the main processor.

Register: FIRMWARE_VERSION

Address: 60004

```
property hardware_installed: int
```

Bitmask indicating installed hardware options. bit0: High Resolution ADC, bit1: WiFi, bit2: RTC, bit3: microSD.

Register: HARDWARE_INSTALLED

Address: 60010

```
property hardware_version: float
```

The hardware version of the device.

Register: HARDWARE_VERSION

Address: 60002

```
property i2c_acks: int
```

A binary encoded value (array of bits) used to observe ACKs from the slave device.

Register: I2C_ACKS

Address: 5114

```
property i2c_data_rx: int
```

Data that has been read from the I2C bus.

This register is a buffer. Underrun behavior - fill with zeros.

Register: I2C_DATA_RX

Address: 5160

```
property i2c_data_tx: int
```

Data that will be written to the I2C bus. This register is a buffer.

Register: I2C_DATA_TX

Address: 5120

```
property i2c_go: int
```

Writing to this register will instruct the LabJack to perform an I2C transaction.

Register: I2C_GO

Address: 5110

```
property i2c_num_bytes_rx: int
```

The number of data bytes to read. Zero is valid and will result in a write-only I2C operation.

Register: I2C_NUM_BYTES_RX

Address: 5109

```
property i2c_num_bytes_tx: int
```

The number of data bytes to transmit. Zero is valid and will result in a read-only I2C operation.

Register: I2C_NUM_BYTES_TX

Address: 5108

```
property i2c_options: int
```

Advanced. Controls details of the I2C protocol to improve device compatibility. bit 0: 1 = Reset the I2C bus before attempting communication. bit 1: 0 = Restarts will use a stop and a start, 1 = Restarts will not use a stop. bit 2: 1 = disable clock stretching.

Register: I2C_OPTIONS

Address: 5103

```
property i2c_scl_dionum: int
```

The number of the DIO line to be used as the I2C clock line. Ex: Writing 1 will force FIO1 to become the I2C-SCL line.

Register: I2C_SCL_DIONUM

Address: 5101

```
property i2c_sda_dionum: int
```

The number of the DIO line to be used as the I2C data line. Ex: Writing 0 will force FIO0 to become the I2C-SDA line.

Register: I2C_SDA_DIONUM

Address: 5100

```
property i2c_slave_address: int
```

The 7-bit address of the slave device. Value is shifted left by firmware to allow room for the I2C R/W bit.

Register: I2C_SLAVE_ADDRESS

Address: 5104

```
property i2c_speed_throttle: int
```

This value controls the I2C clock frequency. Pass 0-65535. Default=0 corresponds to 65536 internally which results in ~450 kHz. 1 results in ~40 Hz, 65516 is ~100 kHz.

Register: I2C_SPEED_THROTTLE

Address: 5102

```
property internal_flash_calculate_crc: int
```

Calculates the checksum of a 4096 byte flash page.

Register: INTERNAL_FLASH_CALCULATE_CRC

Address: 61842

```
property internal_flash_erase: int
```

Erases a 4k section of internal flash starting at the specified address. This register is a buffer.

Register: INTERNAL_FLASH_ERASE

Address: 61820

```
property internal_flash_key: int
```

Sets the region of internal flash to which access is allowed.

Register: INTERNAL_FLASH_KEY

Address: 61800

```
property internal_flash_read: int
```

Data read from internal flash. Read size must be an even number of registers.

Register: INTERNAL_FLASH_READ

Address: 61812

```
property internal_flash_read_pointer: int
```

The address in internal flash that reads will start from.

Register: INTERNAL_FLASH_READ_POINTER

Address: 61810

```
property internal_flash_write: int
```

Data written here will be written to internal flash. Write size must be an even number of registers. This register is a buffer.

Register: INTERNAL_FLASH_WRITE

Address: 61832

```
property internal_flash_write_pointer: int
```

Address in internal flash where writes will begin.

Register: INTERNAL_FLASH_WRITE_POINTER

Address: 61830

```
property io_config_check_for_default: int
```

Returns 1 if the current configuration matches the default configuration.

Register: IO_CONFIG_CHECK_FOR_DEFAULT

Address: 49018

```
property io_config_check_for_factory: int
```

Returns 1 if default settings are the same as factory settings.

Register: IO_CONFIG_CHECK_FOR_FACTORY

Address: 49000

```
property io_config_current_crc32: int
```

Collects the current IO configuration and calculates of a CRC32.

Register: IO_CONFIG_CURRENT_CRC32

Address: 49020

```
property io_config_current_pread: int
```

The address in the current configuration data which reads will start from.

Register: IO_CONFIG_CURRENT_pREAD

Address: 49014

```
property io_config_current_read: int
```

Data read from the current configuration.

Register: IO_CONFIG_CURRENT_READ

Address: 49016

```
property io_config_default_pread: int
```

The address in the configuration data which reads will start from.

Register: IO_CONFIG_DEFAULT_pREAD

Address: 49010

```
property io_config_default_read: int
```

Data read from the startup configuration.

Register: IO_CONFIG_DEFAULT_READ

Address: 49012

```
property io_config_factory_pread: int
```

The address in the factory configuration data which reads will start from.

Register: IO_CONFIG_FACTORY_pREAD

Address: 49006

```
property io_config_factory_read: int
```

Data read from factory configuration.

Register: IO_CONFIG_FACTORY_READ

Address: 49008

```
property io_config_set_current_to_default: int
```

Write a 1 to set current values to default configuration. The default values are retrieved from flash and written to the current configuration registers, thus this behaves similar to reboot/power-up. Systems affected: AIN, DIO, DAC, AIN_EF, DIO_EF.

Register: IO_CONFIG_SET_CURRENT_TO_DEFAULT

Address: 61991

```
property io_config_set_current_to_factory: int
```

Write a 1 to set current values to factory configuration. The factory values are retrieved from flash and written to the current configuration registers. Systems affected: AIN, DIO, DAC, AIN_EF, DIO_EF.

Register: IO_CONFIG_SET_CURRENT_TO_FACTORY

Address: 61990

```
property io_config_set_default_to_current: int
```

Write a 1 to cause new default (reboot/power-up) values to be saved to flash. Current values are retrieved and saved as the new defaults. Systems affected: AIN, DIO, DAC, AIN_EF, DIO_EF.

Register: IO_CONFIG_SET_DEFAULT_TO_CURRENT

Address: 49002

```
property io_config_set_default_to_factory: int
```

Write a 1 to cause new default (reboot/power-up) values to be saved to flash. Factory values are retrieved and saved as the new defaults. Systems affected: AIN, DIO, DAC, AIN_EF, DIO_EF.

Register: IO_CONFIG_SET_DEFAULT_TO_FACTORY

Address: 49004

```
property last_err_detail: int
```

Returns the last LabJack error code seen on the device.

Register: LAST_ERR_DETAIL

Address: 55000

```
property last_err_frame: int
```

Returns the frame number that caused the last error.

Register: LAST_ERR_FRAME

Address: 55002

```
property last_err_transaction_id: int
```

Returns the transaction ID of the Modbus TCP packet that caused the last error.

Register: LAST_ERR_TRANSACTION_ID

Address: 55003

```
property last_mb_err: int
```

Returns the last Modbus TCP error code seen on the device.

Register: LAST_MB_ERR

Address: 55001

```
property led_comm: int
```

Sets the state of the COMM LED when the LEDs are set to manual, see the POWER_LED register.

Register: LED_COMM

Address: 2990

```
property led_status: int
```

Sets the state of the STATUS LED when the LEDs are set to manual, see the POWER_LED register.

Register: LED_STATUS

Address: 2991

```
property lua_debug_data: int
```

Read debug data from here. This register is a buffer.

Register: LUA_DEBUG_DATA

Address: 6024

```
property lua_debug_enable: int
```

Write 1 to this register to enable debugging.

Register: LUA_DEBUG_ENABLE

Address: 6020

```
property lua_debug_enable_default: int
```

This is the value that will be loaded into LUA_DEBUG_ENABLE when the LabJack boots up.

Register: LUA_DEBUG_ENABLE_DEFAULT

Address: 6120

```
property lua_debug_num_bytes: int
```

The number of data bytes in the debug buffer waiting to be read.

Register: LUA_DEBUG_NUM_BYTES

Address: 6022

```
property lua_debug_num_bytes_default: int
```

This is the number of bytes that will be used for the lua debug buffer.

Register: LUA_DEBUG_NUM_BYTES_DEFAULT

Address: 6122

```
property lua_load_saved: int
```

Writing any value reads the script from Flash and loads it into RAM. The script can now be compiled and run.

Register: LUA_LOAD_SAVED

Address: 6034

```
property lua_no_warn_truncation: int
```

Writing a 1 to this register will prevent truncation warnings from being displayed. This register will be cleared every time Lua is started.

Register: LUA_NO_WARN_TRUNCATION

Address: 6050

```
property lua_run: int
```

Writing 1 compiles and runs the Lua script that is loaded in RAM. Writing zero stops the script and begins cleaning up memory. Users may poll the register after writing a value of 0 to verify that the VM is unloaded, and garbage collection is complete. 0 = VM fully unloaded. 1 = Running/in-progress.

Register: LUA_RUN

Address: 6000

```
property lua_run_default: int
```

If set to 1 the script saved in flash will be loaded and run when the LabJack boots up.

Register: LUA_RUN_DEFAULT

Address: 6100

```
property lua_save_to_flash: int
```

Write 1 to save the loaded source code to flash.

Register: LUA_SAVE_TO_FLASH

Address: 6032

```
property lua_saved_read: int
```

Read script saved to flash from here.

Register: LUA_SAVED_READ

Address: 6038

```
property lua_saved_read_pointer: int
```

Address within the saved Lua script in Flash to begin reading from.

Register: LUA_SAVED_READ_POINTER

Address: 6036

```
property lua_source_size: int
```

Allocates RAM for the source code.

Register: LUA_SOURCE_SIZE

Address: 6012

```
property lua_source_write: int
```

Write the source code here. Source will be saved to the RAM allocated by LUA_SOURCE_SIZE. This register is a buffer.

Register: LUA_SOURCE_WRITE

Address: 6014

```
property mio0: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: MIO0 (MIO#(0:2)@0)

Address: 2020

```
property mio1: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: MIO1 (MIO#(0:2)@1)

Address: 2021

```
property mio2: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: MIO2 (MIO#(0:2)@2)

Address: 2022

```
property mio_direction: int
```

Read or write the direction of the 3 bits of MIO in a single binary-encoded value. 0=Input and 1=Output. The upper 8-bits of this value are inhibits.

Register: MIO_DIRECTION

Address: 2603

```
property mio_state: int
```

Read or write the state of the 3 bits of MIO in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. Reading lines set to output returns the current logic levels on the terminals, not necessarily the output states written. The upper 8-bits of this value are inhibits.

Register: MIO_STATE

Address: 2503

```
property onewire_data_rx: int
```

Data received over the 1-wire bus.

This register is a buffer. Underrun behavior - buffer is static, old data will fill the extra locations, firmware 1.0225 changes this to read zeros.

Register: ONEWIRE_DATA_RX

Address: 5370

```
property onewire_data_tx: int
```

Data to be transmitted over the 1-wire bus. This register is a buffer.

Register: ONEWIRE_DATA_TX

Address: 5340

```
property onewire_dpu_dionum: int
```

The dynamic pullup control DIO number.

Register: ONEWIRE_DPU_DIONUM

Address: 5301

```
property onewire_dq_dionum: int
```

The data-line DIO number.

Register: ONEWIRE_DQ_DIONUM

Address: 5300

```
property onewire_function: int
```

Set the ROM function to use. 0xF0=Search, 0xCC=Skip, 0x55=Match, 0x33=Read.

Register: ONEWIRE_FUNCTION

Address: 5307

```
property onewire_go: int
```

Instructs the device to perform the configured 1-wire transaction.

Register: ONEWIRE_GO

Address: 5310

```
property onewire_num_bytes_rx: int
```

Number of data bytes to be received.

Register: ONEWIRE_NUM_BYTES_RX

Address: 5309

```
property onewire_num_bytes_tx: int
```

Number of data bytes to be sent.

Register: ONEWIRE_NUM_BYTES_TX

Address: 5308

```
property onewire_options: int
```

Controls advanced features. Value is a bitmask. bit 0: reserved, bit 1: reserved, bit 2: 1=DPU Enabled 0=DPU Disabled, bit 3: DPU Polarity 1=Active state is high, 0=Active state is low (Dynamic Pull-Up)

Register: ONEWIRE_OPTIONS

Address: 5302

```
property onewire_path_h: int
```

Upper 32-bits of the path to take during a search.

Register: ONEWIRE_PATH_H

Address: 5324

```
property onewire_path_l: int
```

Lower 32-bits of the path to take during a search.

Register: ONEWIRE_PATH_L

Address: 5326

```
property onewire_rom_branchs_found_h: int
```

Upper 32-bits of the branches detected during a search.

Register: ONEWIRE_ROM_BRANCHS_FOUND_H

Address: 5332

```
property onewire_rom_branchs_found_l: int
```

Lower 32-bits of the branches detected during a search.

Register: ONEWIRE_ROM_BRANCHS_FOUND_L

Address: 5334

```
property onewire_rom_match_h: int
```

Upper 32-bits of the ROM to match.

Register: ONEWIRE_ROM_MATCH_H

Address: 5320

```
property onewire_rom_match_l: int
```

Lower 32-bits of the ROM to match.

Register: ONEWIRE_ROM_MATCH_L

Address: 5322

```
property onewire_search_result_h: int
```

Upper 32-bits of the search result.

Register: ONEWIRE_SEARCH_RESULT_H

Address: 5328

```
property onewire_search_result_l: int
```

Lower 32-bites of the search result.

Register: ONEWIRE_SEARCH_RESULT_L

Address: 5330

```
property power_ain: int
```

The current ON/OFF state of the analog input module.

Register: POWER_AIN

Address: 48005

```
property power_ain_default: int
```

The ON/OFF state of the analog input module after a power-cycle to the device. Provided to optionally reduce power consumption.

Register: POWER_AIN_DEFAULT

Address: 48055

```
property power_core: int
```

The core processor speed. 0=80MHz, 1=20MHz, 2=2MHz, 3=250kHz.

Register: POWER_CORE

Address: 48001

```
property power_core_default: int
```

Controls the core processor speed after a power-cycle to the device. 0=80MHz, 1=20MHz, 2=2MHz, 3=250kHz.

Register: POWER_CORE_DEFAULT

Address: 48051

```
property power_ethernet: int
```

The current ON/OFF state of the Ethernet module.

Register: POWER_ETHERNET

Address: 48003

```
property power_ethernet_default: int
```

The ON/OFF state of the Ethernet module after a power-cycle to the device. Provided to optionally reduce power consumption.

Register: POWER_ETHERNET_DEFAULT

Address: 48053

```
property power_led: int
```

Sets the LED operation: 0 = Off. Useful for lower power applications. 1 = normal. 2 = Lower power, LEDs will still blink but will normally be off. 3 = Reserved. 4 = Manual, in this mode the LEDs can be user controlled.

Register: POWER_LED

Address: 48006

```
property power_led_default: int
```

The ON/OFF state of the LEDs after a power-cycle to the device.

Register: POWER_LED_DEFAULT

Address: 48056

```
property power_mode: int
```

The current power management state.

Register: POWER_MODE

Address: 48000

```
property power_mode_default: int
```

The power management state after a power-cycle to the device.

Register: POWER_MODE_DEFAULT

Address: 48050

```
property power_usb: int
```

The current ON/OFF state of the USB module.

Register: POWER_USB

Address: 48002

```
property power_usb_default: int
```

The current ON/OFF state of the USB module after a power-cycle to the device.

Register: POWER_USB_DEFAULT

Address: 48052

```
property power_wifi: int
```

The current ON/OFF state of the WiFi module.

Register: POWER_WIFI

Address: 48004

```
property power_wifi_default: int
```

The ON/OFF state of the WiFi module after a power-cycle to the device. Provided to optionally reduce power consumption.

Register: POWER_WIFI_DEFAULT

Address: 48054

```
property product_id: float
```

The numeric identifier of the device. Such as 7 for a T7 / T7-Pro.

Register: PRODUCT_ID

Address: 60000

```
property rtc_set_time_s: int
```

Write a new timestamp to the RTC in seconds since Jan, 1970, aka Epoch or Unix timestamp.

Register: RTC_SET_TIME_S

Address: 61504

```
property rtc_set_time_sntp: int
```

Write any value to instruct the T7 to update its clock from a SNTP server. Requires that SNTP_UPDATE_INTERVAL is non-zero.

Register: RTC_SET_TIME_SNTP

Address: 61506

```
property rtc_time_calendar: int
```

Read six consecutive addresses of type UINT16, starting with this address. The result will be in year, month, day, hour, minute, second calendar format. i.e. [2014, 10, 21, 18, 55, 35]

Register: RTC_TIME_CALENDAR

Address: 61510

```
property rtc_time_s: int
```

Read the current time in seconds since Jan, 1970, aka Epoch or Unix time. This value is calculated from the 80 MHz crystal, not the RTC 32 kHz crystal. Non-pro devices do not have a real time clock, so the reported time is either seconds since device startup or the time reported by the network time protocol over Ethernet. Pro devices have a real time clock that will be used to initialize this register at startup, the time can then be updated by NTP.

Register: RTC_TIME_S

Address: 61500

```
property sbus0_background_enable: int
```

Currently unsupported.

Register: SBUS0_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@0)

Address: 30250

```
property sbus0_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS0_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@0)

Address: 30225

```
property sbus0_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS0_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@0)

Address: 30200

```
property sbus0_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS0_RH (SBUS#(0:22)_RH@0)

Address: 30150

```
property sbus0_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS0_TEMP (SBUS#(0:22)_TEMP@0)

Address: 30100

```
property sbus10_background_enable: int
```

Currently unsupported.

Register: SBUS10_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@10)

Address: 30260

```
property sbus10_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS10_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@10)

Address: 30235

```
property sbus10_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS10_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@10)

Address: 30210

```
property sbus10_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS10_RH (SBUS#(0:22)_RH@10)

Address: 30170

```
property sbus10_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS10_TEMP (SBUS#(0:22)_TEMP@10)

Address: 30120

```
property sbus11_background_enable: int
```

Currently unsupported.

Register: SBUS11_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@11)

Address: 30261

```
property sbus11_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS11_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@11)

Address: 30236

```
property sbus11_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS11_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@11)

Address: 30211

```
property sbus11_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS11_RH (SBUS#(0:22)_RH@11)

Address: 30172

```
property sbus11_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS11_TEMP (SBUS#(0:22)_TEMP@11)

Address: 30122

```
property sbus12_background_enable: int
```

Currently unsupported.

Register: SBUS12_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@12)

Address: 30262

```
property sbus12_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS12_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@12)

Address: 30237

```
property sbus12_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS12_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@12)

Address: 30212

```
property sbus12_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS12_RH (SBUS#(0:22)_RH@12)

Address: 30174

```
property sbus12_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS12_TEMP (SBUS#(0:22)_TEMP@12)

Address: 30124

```
property sbus13_background_enable: int
```

Currently unsupported.

Register: SBUS13_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@13)

Address: 30263

```
property sbus13_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS13_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@13)

Address: 30238

```
property sbus13_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS13_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@13)

Address: 30213

```
property sbus13_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS13_RH (SBUS#(0:22)_RH@13)

Address: 30176

```
property sbus13_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS13_TEMP (SBUS#(0:22)_TEMP@13)

Address: 30126

```
property sbus14_background_enable: int
```

Currently unsupported.

Register: SBUS14_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@14)

Address: 30264

```
property sbus14_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS14_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@14)

Address: 30239

```
property sbus14_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS14_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@14)

Address: 30214

```
property sbus14_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS14_RH (SBUS#(0:22)_RH@14)

Address: 30178

```
property sbus14_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS14_TEMP (SBUS#(0:22)_TEMP@14)

Address: 30128

```
property sbus15_background_enable: int
```

Currently unsupported.

Register: SBUS15_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@15)

Address: 30265

```
property sbus15_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS15_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@15)

Address: 30240

```
property sbus15_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS15_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@15)

Address: 30215

```
property sbus15_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS15_RH (SBUS#(0:22)_RH@15)

Address: 30180

```
property sbus15_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS15_TEMP (SBUS#(0:22)_TEMP@15)

Address: 30130

```
property sbus16_background_enable: int
```

Currently unsupported.

Register: SBUS16_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@16)

Address: 30266

```
property sbus16_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS16_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@16)

Address: 30241

```
property sbus16_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS16_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@16)

Address: 30216

```
property sbus16_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS16_RH (SBUS#(0:22)_RH@16)

Address: 30182

```
property sbus16_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS16_TEMP (SBUS#(0:22)_TEMP@16)

Address: 30132

```
property sbus17_background_enable: int
```

Currently unsupported.

Register: SBUS17_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@17)

Address: 30267

```
property sbus17_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS17_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@17)

Address: 30242

```
property sbus17_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS17_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@17)

Address: 30217

```
property sbus17_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS17_RH (SBUS#(0:22)_RH@17)

Address: 30184

```
property sbus17_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS17_TEMP (SBUS#(0:22)_TEMP@17)

Address: 30134

```
property sbus18_background_enable: int
```

Currently unsupported.

Register: SBUS18_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@18)

Address: 30268

```
property sbus18_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS18_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@18)

Address: 30243

```
property sbus18_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS18_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@18)

Address: 30218

```
property sbus18_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS18_RH (SBUS#(0:22)_RH@18)

Address: 30186

```
property sbus18_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS18_TEMP (SBUS#(0:22)_TEMP@18)

Address: 30136

```
property sbus19_background_enable: int
```

Currently unsupported.

Register: SBUS19_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@19)

Address: 30269

```
property sbus19_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS19_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@19)

Address: 30244

```
property sbus19_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS19_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@19)

Address: 30219

```
property sbus19_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS19_RH (SBUS#(0:22)_RH@19)

Address: 30188

```
property sbus19_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS19_TEMP (SBUS#(0:22)_TEMP@19)

Address: 30138

```
property sbus1_background_enable: int
```

Currently unsupported.

Register: SBUS1_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@1)

Address: 30251

```
property sbus1_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS1_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@1)

Address: 30226

```
property sbus1_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS1_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@1)

Address: 30201

```
property sbus1_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS1_RH (SBUS#(0:22)_RH@1)

Address: 30152

```
property sbus1_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS1_TEMP (SBUS#(0:22)_TEMP@1)

Address: 30102

```
property sbus20_background_enable: int
```

Currently unsupported.

Register: SBUS20_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@20)

Address: 30270

```
property sbus20_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS20_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@20)

Address: 30245

```
property sbus20_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS20_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@20)

Address: 30220

```
property sbus20_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS20_RH (SBUS#(0:22)_RH@20)

Address: 30190

```
property sbus20_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS20_TEMP (SBUS#(0:22)_TEMP@20)

Address: 30140

```
property sbus21_background_enable: int
```

Currently unsupported.

Register: SBUS21_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@21)

Address: 30271

```
property sbus21_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS21_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@21)

Address: 30246

```
property sbus21_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS21_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@21)

Address: 30221

```
property sbus21_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS21_RH (SBUS#(0:22)_RH@21)

Address: 30192

```
property sbus21_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS21_TEMP (SBUS#(0:22)_TEMP@21)

Address: 30142

```
property sbus22_background_enable: int
```

Currently unsupported.

Register: SBUS22_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@22)

Address: 30272

```
property sbus22_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS22_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@22)

Address: 30247

```
property sbus22_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS22_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@22)

Address: 30222

```
property sbus22_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS22_RH (SBUS#(0:22)_RH@22)

Address: 30194

```
property sbus22_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS22_TEMP (SBUS#(0:22)_TEMP@22)

Address: 30144

```
property sbus2_background_enable: int
```

Currently unsupported.

Register: SBUS2_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@2)

Address: 30252

```
property sbus2_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS2_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@2)

Address: 30227

```
property sbus2_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS2_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@2)

Address: 30202

```
property sbus2_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS2_RH (SBUS#(0:22)_RH@2)

Address: 30154

```
property sbus2_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS2_TEMP (SBUS#(0:22)_TEMP@2)

Address: 30104

```
property sbus3_background_enable: int
```

Currently unsupported.

Register: SBUS3_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@3)

Address: 30253

```
property sbus3_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS3_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@3)

Address: 30228

```
property sbus3_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS3_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@3)

Address: 30203

```
property sbus3_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS3_RH (SBUS#(0:22)_RH@3)

Address: 30156

```
property sbus3_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS3_TEMP (SBUS#(0:22)_TEMP@3)

Address: 30106

```
property sbus4_background_enable: int
```

Currently unsupported.

Register: SBUS4_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@4)

Address: 30254

```
property sbus4_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS4_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@4)

Address: 30229

```
property sbus4_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS4_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@4)

Address: 30204

```
property sbus4_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS4_RH (SBUS#(0:22)_RH@4)

Address: 30158

```
property sbus4_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS4_TEMP (SBUS#(0:22)_TEMP@4)

Address: 30108

```
property sbus5_background_enable: int
```

Currently unsupported.

Register: SBUS5_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@5)

Address: 30255

```
property sbus5_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS5_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@5)

Address: 30230

```
property sbus5_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS5_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@5)

Address: 30205

```
property sbus5_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS5_RH (SBUS#(0:22)_RH@5)

Address: 30160

```
property sbus5_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS5_TEMP (SBUS#(0:22)_TEMP@5)

Address: 30110

```
property sbus6_background_enable: int
```

Currently unsupported.

Register: SBUS6_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@6)

Address: 30256

```
property sbus6_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS6_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@6)

Address: 30231

```
property sbus6_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS6_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@6)

Address: 30206

```
property sbus6_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS6_RH (SBUS#(0:22)_RH@6)

Address: 30162

```
property sbus6_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS6_TEMP (SBUS#(0:22)_TEMP@6)

Address: 30112

```
property sbus7_background_enable: int
```

Currently unsupported.

Register: SBUS7_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@7)

Address: 30257

```
property sbus7_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS7_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@7)

Address: 30232

```
property sbus7_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS7_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@7)

Address: 30207

```
property sbus7_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS7_RH (SBUS#(0:22)_RH@7)

Address: 30164

```
property sbus7_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS7_TEMP (SBUS#(0:22)_TEMP@7)

Address: 30114

```
property sbus8_background_enable: int
```

Currently unsupported.

Register: SBUS8_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@8)

Address: 30258

```
property sbus8_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS8_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@8)

Address: 30233

```
property sbus8_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS8_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@8)

Address: 30208

```
property sbus8_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS8_RH (SBUS#(0:22)_RH@8)

Address: 30166

```
property sbus8_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS8_TEMP (SBUS#(0:22)_TEMP@8)

Address: 30116

```
property sbus9_background_enable: int
```

Currently unsupported.

Register: SBUS9_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@9)

Address: 30259

```
property sbus9_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS9_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@9)

Address: 30234

```
property sbus9_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS9_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@9)

Address: 30209

```
property sbus9_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS9_RH (SBUS#(0:22)_RH@9)

Address: 30168

```
property sbus9_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS9_TEMP (SBUS#(0:22)_TEMP@9)

Address: 30118

```
property sbus_all_clock_dionum: int
```

A write to this global parameter sets all SBUS clock line registers to the same value. A read will return the correct setting if all channels are set the same, but otherwise will return 0xFF.

Register: SBUS_ALL_CLOCK_DIONUM

Address: 30276

```
property sbus_all_clock_speed: int
```

Sets the clock speed. The clock is software generated so the resulting frequency is not exact. Valid range is 0-65535. Larger values are faster. 0 is the fastest option and is equivalent to 65536. A value of 0 is ~200 kHz. A value of 65000 is ~9.1 kHz.

Register: SBUS_ALL_CLOCK_SPEED

Address: 30278

```
property sbus_all_data_dionum: int
```

A write to this global parameter sets all SBUS data line registers to the same value. A read will return the correct setting if all channels are set the same, but otherwise will return 0xFF.

Register: SBUS_ALL_DATA_DIONUM

Address: 30275

```
property sbus_all_power_dionum: int
```

Sets the power line. This DIO is set to output-high upon any read of SBUS#_TEMP or SBUS#_RH. Default is FIO6 for the T4 and FIO2 for the T7. An FIO line can power up to 4 sensors while an EIO/CIO/MIO line or DAC line can power up to 20 sensors. Set to 9999 to disable. To use multiple power lines, use a DAC line for power, or otherwise control power yourself, set this to 9999 and then control power using writes to normal registers such as FIO5, EIO0, or DAC0.

Register: SBUS_ALL_POWER_DIONUM

Address: 30277

```
property serial_number: int
```

The serial number of the device.

Register: SERIAL_NUMBER

Address: 60028

```
setup_ain0(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN0.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain

- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain1(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN1.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain10(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN10.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain100(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup AIN100.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain101(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup AIN101.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain102(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup AIN102.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain103(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup AIN103.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain104(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup AIN104.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain105(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup AIN105.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain106(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup AIN106.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain107(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup AIN107.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain108(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup AIN108.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain109(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup AIN109.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain11(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN11.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain

- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain110(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup AIN110.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain111(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup AIN111.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain112(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup AIN112.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain113(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup AIN113.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain114(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup AIN114.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain115(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup AIN115.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain116(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup AIN116.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain117(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup AIN117.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain118(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup AIN118.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain119(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup AIN119.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain12(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN12.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain120(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup AIN120.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain121(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup AIN121.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain

- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain122(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup AIN122.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain123(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup AIN123.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain124(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup AIN124.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain125(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup AIN125.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain126(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup AIN126.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain127(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup AIN127.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain13(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN13.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain2(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN2.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain3(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN3.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain4(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN4.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain48(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN48.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain49(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN49.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain5(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN5.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain

- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain50(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN50.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain51(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN51.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain52(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN52.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain53(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN53.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain54(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN54.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain55(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN55.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain56(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN56.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain57(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN57.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain58(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN58.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain59(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN59.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain6(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN6.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain60(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN60.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain61(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN61.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain

- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain62(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN62.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain63(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN63.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain64(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN64.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain65(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN65.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain66(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN66.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain67(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN67.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain68(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN68.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain69(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN69.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain7(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN7.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain70(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN70.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain71(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN71.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain72(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN72.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain73(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN73.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain

- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain74(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN74.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain75(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN75.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain76(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN76.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain77(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN77.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain78(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN78.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain79(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN79.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain8(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN8.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain80(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN80.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain81(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN81.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain82(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN82.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain83(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN83.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain84(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN84.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain85(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN85.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain

- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain86(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN86.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain87(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN87.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain88(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN88.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain89(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN89.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain9(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN9.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain90(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN90.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain91(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN91.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain92(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN92.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain93(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN93.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain94(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN94.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain95(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN95.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain96(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN96.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain97(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN97.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain

- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain98(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN98.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain99(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN99.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain_all(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup all AIN channels.

```
property sntp_update_interval: int
```

Sets the SNTP retry time in seconds. A value of zero will disable SNTP(0=default). Values must be 10 or greater.

Register: SNTP_UPDATE_INTERVAL

Address: 49702

```
property spc_frequency_out_enable_deprecated: int
```

This has been deprecated in favor of DAC1_FREQUENCY_OUT_ENABLE. 0 = off, 1 = output 10 Hz signal on SPC. Note that stream uses SPC for a timing output, so if you use this while streaming you will get a lot more counts.

Register: SPC_FREQUENCY_OUT_ENABLE_DEPRECATED

Address: 61530

```
property spi_clk_dionum: int
```

The DIO line for Clock.

Register: SPI_CLK_DIONUM

Address: 5001

```
property spi_cs_dionum: int
```

The DIO line for Chip-Select.

Register: SPI_CS_DIONUM

Address: 5000

```
property spi_data_rx: int
```

Read data here.

This register is a buffer. Underrun behavior - fill with zeros.

Register: SPI_DATA_RX

Address: 5050

```
property spi_data_tx: int
```

Write data here. This register is a buffer.

Register: SPI_DATA_TX

Address: 5010

```
property spi_go: int
```

Write 1 to begin the configured SPI transaction.

Register: SPI_GO

Address: 5007

```
property spi_miso_dionum: int
```

The DIO line for Master-In-Slave-Out.

Register: SPI_MISO_DIONUM

Address: 5002

```
property spi_mode: int
```

The SPI mode controls the clock idle state and which edge clocks the data. Bit 1 is CPOL and Bit 0 is CPHA, so CPOL/CPHA for different decimal values: 0 = 0/0 = b00, 1 = 0/1 = b01, 2 = 1/0 = b10, 3 = 1/1 = b11. For CPOL and CPHA explanations, see Wikipedia article: https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus.

Register: SPI_MODE

Address: 5004

```
property spi_mosi_dionum: int
```

The DIO line for Master-Out-Slave-In.

Register: SPI_MOSI_DIONUM

Address: 5003

```
property spi_num_bytes: int
```

The number of bytes to transfer. The maximum transfer size is 100 bytes.

Register: SPI_NUM_BYTES

Address: 5009

```
property spi_options: int
```

Bit 0 is Auto-CS-Disable. When bit 0 is 0, CS is enabled. When bit 0 is 1, CS is disabled. Bit 1: 0 = Set DIO directions before starting the SPI operations, 1 = Do not set DIO directions. Bit 2: 0 = Transmit data MSB first, 1 = LSB first. Bits 4-7: This value sets the number of bits that will be transmitted during the last byte of the SPI operation. Default is 8, valid options are 1-8.

Register: SPI_OPTIONS

Address: 5006

```
property spi_speed_throttle: int
```

This value controls the SPI clock frequency. Pass 0-65535. Default=0 corresponds to 65536 internally which results in ~800 kHz. 65500 = ~100 kHz, 65100 = ~10 kHz, 61100 = ~1 kHz, 21000 = ~100 Hz, and 1 = ~67 Hz. Avoid setting too low such that the entire transaction lasts longer than the 250 millisecond timeout of the internal watchdog timer.

Register: SPI_SPEED_THROTTLE

Address: 5005

```
property stream_auto_target: int
```

Controls where data will be sent. Value is a bitmask. bit 0: 1 = Send to Ethernet 702 sockets, bit 1: 1 = Send to USB, bit 4: 1 = Command-Response mode. All other bits are reserved.

Register: STREAM_AUTO_TARGET

Address: 4016

```
property stream_autorecover_disable: int
```

When the stream buffer becomes full the stream will be stopped. Same behavior as if STREAM_ENABLE is set to zero.

Register: STREAM_AUTORECOVER_DISABLE

Address: 4028

```
property stream_buffer_size_bytes: int
```

Size of the stream data buffer in bytes. A value of 0 equates to the default value. Must be a power of 2. Size in samples is STREAM_BUFFER_SIZE_BYTES/2. Size in scans is (STREAM_BUFFER_SIZE_BYTES/2)/STREAM_NUM_ADDRESSES. Changes while stream is running do not affect the currently running stream.

Register: STREAM_BUFFER_SIZE_BYTES

Address: 4012

```
property stream_clock_source: int
```

Controls which clock source will be used to run the main stream clock. 0 = Internal crystal, 2 = External clock source on CIO3. Rising edges will increment a counter and trigger a stream scan after the number of edges specified in STREAM_EXTERNAL_CLOCK_DIVISOR. T7 will expect external stream clock on CIO3. All other values reserved.

Register: STREAM_CLOCK_SOURCE

Address: 4014

```
property stream_data_capture_16: int
```

If a channel produces 32-bits of data the upper 16 will be saved here.

Register: STREAM_DATA_CAPTURE_16

Address: 4899

```
property stream_data_cr: int
```

Address to read stream data from when operating in C-R mode.

Register: STREAM_DATA_CR

Address: 4500

```
property stream_debug_get_self_index: int
```

Returns the index of the channel. The index is the position in the scan list that the channel occupies.

Register: STREAM_DEBUG_GET_SELF_INDEX

Address: 4898

```
property stream_enable: int
```

Write 1 to start stream. Write 0 to stop stream. Reading this register returns 1 when stream is enabled. When using a triggered stream the stream is considered enabled while waiting for the trigger.

Register: STREAM_ENABLE

Address: 4990

```
property stream_external_clock_divisor: int
```

The number of pulses per stream scan when using an external clock.

Register: STREAM_EXTERNAL_CLOCK_DIVISOR

Address: 4022

```
property stream_num_addresses: int
```

The number of entries in the scanlist.

Register: STREAM_NUM_ADDRESSES

Address: 4004

```
property stream_num_scans: int
```

The number of scans to run before automatically stopping (stream-burst). 0 = run continuously. Limit for STREAM_NUM_SCANS is 2^32-1, but if the host is not reading data as fast as it is acquired you also need to consider STREAM_BUFFER_SIZE_BYTES.

Register: STREAM_NUM_SCANS

Address: 4020

```
property stream_out0: int
```

127) to trigger stream-out updates. When added to the scan list these do count against the max scan rate just like normal input addresses, but they do not return any data in the stream read.

Register: STREAM_OUT0 (STREAM_OUT#(0:3)@0)

Address: 4800

Type:
Include one or more of these registers in STREAM_SCANLIST_ADDRESS#(0

```
property stream_out0_buffer_allocate_num_bytes: int
```

Size of the buffer in bytes as a power of 2. Should be at least twice the size of updates that will be written and no less than 32. The usable buffer size will be equal to the value of this register minus 2 bytes (one 16-bit sample). Before writing data to _BUFFER_###, you must write to _BUFFER_ALLOCATE_NUM_BYTES to allocate RAM for the data. Max is 16384.

Register: STREAM_OUT0_BUFFER_ALLOCATE_NUM_BYTES
(STREAM_OUT#(0:3)_BUFFER_ALLOCATE_NUM_BYTES@0)

Address: 4050

```
property stream_out0_buffer_f32: float
```

Data destination when sending floating point data. Appropriate cal constants are used to convert F32 values to 16-bit binary data, and thus each of these values uses 2 bytes of the stream-out buffer. This register is a buffer.

Register: STREAM_OUT0_BUFFER_F32 (STREAM_OUT#(0:3)_BUFFER_F32@0)

Address: 4400

```
property stream_out0_buffer_status: int
```

The number of values in the buffer that are not currently being used.

Register: STREAM_OUT0_BUFFER_STATUS (STREAM_OUT#(0:3)_BUFFER_STATUS@0)

Address: 4080

```
property stream_out0_buffer_u16: int
```

Data destination when sending 16-bit integer data. Each value uses 2 bytes of the stream-out buffer. This register is a buffer.

Register: STREAM_OUT0_BUFFER_U16 (STREAM_OUT#(0:3)_BUFFER_U16@0)

Address: 4420

```
property stream_out0_buffer_u32: int
```

Not used at this time. There are no U32 registers supported in stream-out. This register is a buffer.

Register: STREAM_OUT0_BUFFER_U32 (STREAM_OUT#(0:3)_BUFFER_U32@0)

Address: 4410

```
property stream_out0_enable: int
```

When STREAM_OUT#_ENABLE is enabled, the stream out target is generally updated by one value from the stream out buffer per stream scan. For example, there will generally be one instance of e.g. STREAM_OUT0 in the stream scan list, which will cause one STREAM_OUT0_BUFFER value to be consumed and written to STREAM_OUT0_TARGET for every stream scan. The stream scan list could also contain two instances of STREAM_OUT0, in which case two values from STREAM_OUT0_BUFFER value would be consumed and written for every stream scan.

Register: STREAM_OUT0_ENABLE (STREAM_OUT#(0:3)_ENABLE@0)

Address: 4090

```
property stream_out0_loop_num_values: int
```

The number of values, from the end of the array, that will be repeated after reaching the end of supplied data array.

Register: STREAM_OUT0_LOOP_NUM_VALUES (STREAM_OUT#(0:3)_LOOP_NUM_VALUES@0)

Address: 4060

```
property stream_out0_set_loop: int
```

Controls when new data and loop size are used. 1=Use new data immediately. 2=Wait for synch. New data will not be used until a different stream-out channel is set to Synch. 3=Synch. This stream-out# as well as any stream-outs set to synch will start using new data immediately.

Register: STREAM_OUT0_SET_LOOP (STREAM_OUT#(0:3)_SET_LOOP@0)

Address: 4070

```
property stream_out0_target: int
```

Channel that data will be written to. Before writing data to _BUFFER_###, you must write to _TARGET so the device knows how to interpret and store values.

Register: STREAM_OUT0_TARGET (STREAM_OUT#(0:3)_TARGET@0)

Address: 4040

```
property stream_out1: int
```

127) to trigger stream-out updates. When added to the scan list these do count against the max scan rate just like normal input addresses, but they do not return any data in the stream read.

Register: STREAM_OUT1 (STREAM_OUT#(0:3)@1)

Address: 4801

Type:
Include one or more of these registers in STREAM_SCANLIST_ADDRESS#(0

```
property stream_out1_buffer_allocate_num_bytes: int
```

Size of the buffer in bytes as a power of 2. Should be at least twice the size of updates that will be written and no less than 32. The usable buffer size will be equal to the value of this register minus 2 bytes (one 16-bit sample). Before writing data to _BUFFER_###, you must write to _BUFFER_ALLOCATE_NUM_BYTES to allocate RAM for the data. Max is 16384.

Register: STREAM_OUT1_BUFFER_ALLOCATE_NUM_BYTES (STREAM_OUT#(0:3)_BUFFER_ALLOCATE_NUM_BYTES@1)

Address: 4052

```
property stream_out1_buffer_f32: float
```

Data destination when sending floating point data. Appropriate cal constants are used to convert F32 values to 16-bit binary data, and thus each of these values uses 2 bytes of the stream-out buffer. This register is a buffer.

Register: STREAM_OUT1_BUFFER_F32 (STREAM_OUT#(0:3)_BUFFER_F32@1)

Address: 4402

```
property stream_out1_buffer_status: int
```

The number of values in the buffer that are not currently being used.

Register: STREAM_OUT1_BUFFER_STATUS (STREAM_OUT#(0:3)_BUFFER_STATUS@1)

Address: 4082

```
property stream_out1_buffer_u16: int
```

Data destination when sending 16-bit integer data. Each value uses 2 bytes of the stream-out buffer. This register is a buffer.

Register: STREAM_OUT1_BUFFER_U16 (STREAM_OUT#(0:3)_BUFFER_U16@1)

Address: 4421

```
property stream_out1_buffer_u32: int
```

Not used at this time. There are no U32 registers supported in stream-out. This register is a buffer.

Register: STREAM_OUT1_BUFFER_U32 (STREAM_OUT#(0:3)_BUFFER_U32@1)

Address: 4412

```
property stream_out1_enable: int
```

When STREAM_OUT#_ENABLE is enabled, the stream out target is generally updated by one value from the stream out buffer per stream scan. For example, there will generally be one instance of e.g. STREAM_OUT0 in the stream scan list, which will cause one

STREAM_OUT0_BUFFER value to be consumed and written to STREAM_OUT0_TARGET for every stream scan. The stream scan list could also contain two instances of STREAM_OUT0, in which case two values from STREAM_OUT0_BUFFER value would be consumed and written for every stream scan.

Register: STREAM_OUT1_ENABLE (STREAM_OUT#(0:3)_ENABLE@1)

Address: 4092

```
property stream_out1_loop_num_values: int
```

The number of values, from the end of the array, that will be repeated after reaching the end of supplied data array.

Register: STREAM_OUT1_LOOP_NUM_VALUES (STREAM_OUT#(0:3)_LOOP_NUM_VALUES@1)

Address: 4062

```
property stream_out1_set_loop: int
```

Controls when new data and loop size are used. 1=Use new data immediately. 2=Wait for synch. New data will not be used until a different stream-out channel is set to Synch. 3=Synch. This stream-out# as well as any stream-outs set to synch will start using new data immediately.

Register: STREAM_OUT1_SET_LOOP (STREAM_OUT#(0:3)_SET_LOOP@1)

Address: 4072

```
property stream_out1_target: int
```

Channel that data will be written to. Before writing data to _BUFFER_###, you must write to _TARGET so the device knows how to interpret and store values.

Register: STREAM_OUT1_TARGET (STREAM_OUT#(0:3)_TARGET@1)

Address: 4042

```
property stream_out2: int
```

127) to trigger stream-out updates. When added to the scan list these do count against the max scan rate just like normal input addresses, but they do not return any data in the stream read.

Register: STREAM_OUT2 (STREAM_OUT#(0:3)@2)

Address: 4802

Type:
Include one or more of these registers in STREAM_SCANLIST_ADDRESS#(0

```
property stream_out2_buffer_allocate_num_bytes: int
```

Size of the buffer in bytes as a power of 2. Should be at least twice the size of updates that will be written and no less than 32. The usable buffer size will be equal to the value of this register minus 2 bytes (one 16-bit sample). Before writing data to _BUFFER_###, you must write to _BUFFER_ALLOCATE_NUM_BYTES to allocate RAM for the data. Max is 16384.

Register: STREAM_OUT2_BUFFER_ALLOCATE_NUM_BYTES (STREAM_OUT#(0:3)_BUFFER_ALLOCATE_NUM_BYTES@2)

Address: 4054

```
property stream_out2_buffer_f32: float
```

Data destination when sending floating point data. Appropriate cal constants are used to convert F32 values to 16-bit binary data, and thus each of these values uses 2 bytes of the stream-out buffer. This register is a buffer.

Register: STREAM_OUT2_BUFFER_F32 (STREAM_OUT#(0:3)_BUFFER_F32@2)

Address: 4404

```
property stream_out2_buffer_status: int
```

The number of values in the buffer that are not currently being used.

Register: STREAM_OUT2_BUFFER_STATUS (STREAM_OUT#(0:3)_BUFFER_STATUS@2)

Address: 4084

```
property stream_out2_buffer_u16: int
```

Data destination when sending 16-bit integer data. Each value uses 2 bytes of the stream-out buffer. This register is a buffer.

Register: STREAM_OUT2_BUFFER_U16 (STREAM_OUT#(0:3)_BUFFER_U16@2)

Address: 4422

```
property stream_out2_buffer_u32: int
```

Not used at this time. There are no U32 registers supported in stream-out. This register is a buffer.

Register: STREAM_OUT2_BUFFER_U32 (STREAM_OUT#(0:3)_BUFFER_U32@2)

Address: 4414

```
property stream_out2_enable: int
```

When STREAM_OUT#_ENABLE is enabled, the stream out target is generally updated by one value from the stream out buffer per stream scan. For example, there will generally be one instance of e.g. STREAM_OUT0 in the stream scan list, which will cause one STREAM_OUT0_BUFFER value to be consumed and written to STREAM_OUT0_TARGET for every stream scan. The stream scan list could also contain two instances of STREAM_OUT0, in which case two values from STREAM_OUT0_BUFFER value would be consumed and written for every stream scan.

Register: STREAM_OUT2_ENABLE (STREAM_OUT#(0:3)_ENABLE@2)

Address: 4094

```
property stream_out2_loop_num_values: int
```

The number of values, from the end of the array, that will be repeated after reaching the end of supplied data array.

Register: STREAM_OUT2_LOOP_NUM_VALUES (STREAM_OUT#(0:3)_LOOP_NUM_VALUES@2)

Address: 4064

```
property stream_out2_set_loop: int
```

Controls when new data and loop size are used. 1=Use new data immediately. 2=Wait for synch. New data will not be used until a different stream-out channel is set to Synch. 3=Synch. This stream-out# as well as any stream-outs set to synch will start using new data immediately.

Register: STREAM_OUT2_SET_LOOP (STREAM_OUT#(0:3)_SET_LOOP@2)

Address: 4074

```
property stream_out2_target: int
```

Channel that data will be written to. Before writing data to _BUFFER_###, you must write to _TARGET so the device knows how to interpret and store values.

Register: STREAM_OUT2_TARGET (STREAM_OUT#(0:3)_TARGET@2)

Address: 4044

```
property stream_out3: int
```

127) to trigger stream-out updates. When added to the scan list these do count against the max scan rate just like normal input addresses, but they do not return any data in the stream read.

Register: STREAM_OUT3 (STREAM_OUT#(0:3)@3)

Address: 4803

Type:
Include one or more of these registers in STREAM_SCANLIST_ADDRESS#(0

```
property stream_out3_buffer_allocate_num_bytes: int
```

Size of the buffer in bytes as a power of 2. Should be at least twice the size of updates that will be written and no less than 32. The usable buffer size will be equal to the value of this register minus 2 bytes (one 16-bit sample). Before writing data to _BUFFER_###, you must write to _BUFFER_ALLOCATE_NUM_BYTES to allocate RAM for the data. Max is 16384.

Register: STREAM_OUT3_BUFFER_ALLOCATE_NUM_BYTES
(STREAM_OUT#(0:3)_BUFFER_ALLOCATE_NUM_BYTES@3)

Address: 4056

```
property stream_out3_buffer_f32: float
```

Data destination when sending floating point data. Appropriate cal constants are used to convert F32 values to 16-bit binary data, and thus each of these values uses 2 bytes of the stream-out buffer. This register is a buffer.

Register: STREAM_OUT3_BUFFER_F32 (STREAM_OUT#(0:3)_BUFFER_F32@3)

Address: 4406

```
property stream_out3_buffer_status: int
```

The number of values in the buffer that are not currently being used.

Register: STREAM_OUT3_BUFFER_STATUS (STREAM_OUT#(0:3)_BUFFER_STATUS@3)

Address: 4086

```
property stream_out3_buffer_u16: int
```

Data destination when sending 16-bit integer data. Each value uses 2 bytes of the stream-out buffer. This register is a buffer.

Register: STREAM_OUT3_BUFFER_U16 (STREAM_OUT#(0:3)_BUFFER_U16@3)

Address: 4423

```
property stream_out3_buffer_u32: int
```

Not used at this time. There are no U32 registers supported in stream-out. This register is a buffer.

Register: STREAM_OUT3_BUFFER_U32 (STREAM_OUT#(0:3)_BUFFER_U32@3)

Address: 4416

```
property stream_out3_enable: int
```

When STREAM_OUT#_ENABLE is enabled, the stream out target is generally updated by one value from the stream out buffer per stream scan. For example, there will generally be one instance of e.g. STREAM_OUT0 in the stream scan list, which will cause one STREAM_OUT0_BUFFER value to be consumed and written to STREAM_OUT0_TARGET for every stream scan. The stream scan list could also contain two instances of STREAM_OUT0, in which case two values from STREAM_OUT0_BUFFER value would be consumed and written for every stream scan.

Register: STREAM_OUT3_ENABLE (STREAM_OUT#(0:3)_ENABLE@3)

Address: 4096

```
property stream_out3_loop_num_values: int
```

The number of values, from the end of the array, that will be repeated after reaching the end of supplied data array.

Register: STREAM_OUT3_LOOP_NUM_VALUES (STREAM_OUT#(0:3)_LOOP_NUM_VALUES@3)

Address: 4066

```
property stream_out3_set_loop: int
```

Controls when new data and loop size are used. 1=Use new data immediately. 2=Wait for synch. New data will not be used until a different stream-out channel is set to Synch. 3=Synch. This stream-out# as well as any stream-outs set to synch will start using new data immediately.

Register: STREAM_OUT3_SET_LOOP (STREAM_OUT#(0:3)_SET_LOOP@3)

Address: 4076

```
property stream_out3_target: int
```

Channel that data will be written to. Before writing data to _BUFFER_###, you must write to _TARGET so the device knows how to interpret and store values.

Register: STREAM_OUT3_TARGET (STREAM_OUT#(0:3)_TARGET@3)

Address: 4046

```
property stream_resolution_index: int
```

The resolution index for stream readings. A larger resolution index generally results in lower noise and longer sample times.

Register: STREAM_RESOLUTION_INDEX

Address: 4010

```
property stream_samples_per_packet: int
```

Specifies the number of data points to be sent in the data packet. Only applies to spontaneous mode.

Register: STREAM_SAMPLES_PER_PACKET

Address: 4006

```
property stream_scanlist_address0: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS0 (STREAM_SCANLIST_ADDRESS#(0:127)@0)

Address: 4100

```
property stream_scanlist_address1: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS1 (STREAM_SCANLIST_ADDRESS#(0:127)@1)

Address: 4102

```
property stream_scanlist_address10: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS10 (STREAM_SCANLIST_ADDRESS#(0:127)@10)

Address: 4120

```
property stream_scanlist_address100: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS100 (STREAM_SCANLIST_ADDRESS#(0:127)@100)

Address: 4300

```
property stream_scanlist_address101: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS101 (STREAM_SCANLIST_ADDRESS#(0:127)@101)

Address: 4302

```
property stream_scanlist_address102: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS102 (STREAM_SCANLIST_ADDRESS#(0:127)@102)

Address: 4304

```
property stream_scanlist_address103: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS103 (STREAM_SCANLIST_ADDRESS#(0:127)@103)

Address: 4306

```
property stream_scanlist_address104: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS104 (STREAM_SCANLIST_ADDRESS#(0:127)@104)

Address: 4308

```
property stream_scanlist_address105: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS105 (STREAM_SCANLIST_ADDRESS#(0:127)@105)

Address: 4310

```
property stream_scanlist_address106: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS106 (STREAM_SCANLIST_ADDRESS#(0:127)@106)

Address: 4312

```
property stream_scanlist_address107: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS107 (STREAM_SCANLIST_ADDRESS#(0:127)@107)

Address: 4314

```
property stream_scanlist_address108: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS108 (STREAM_SCANLIST_ADDRESS#(0:127)@108)

Address: 4316

```
property stream_scanlist_address109: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS109 (STREAM_SCANLIST_ADDRESS#(0:127)@109)

Address: 4318

```
property stream_scanlist_address11: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS11 (STREAM_SCANLIST_ADDRESS#(0:127)@11)

Address: 4122

```
property stream_scanlist_address110: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS110 (STREAM_SCANLIST_ADDRESS#(0:127)@110)

Address: 4320

```
property stream_scanlist_address111: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS111 (STREAM_SCANLIST_ADDRESS#(0:127)@111)

Address: 4322

```
property stream_scanlist_address112: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS112 (STREAM_SCANLIST_ADDRESS#(0:127)@112)

Address: 4324

```
property stream_scanlist_address113: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS113 (STREAM_SCANLIST_ADDRESS#(0:127)@113)

Address: 4326

```
property stream_scanlist_address114: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS114 (STREAM_SCANLIST_ADDRESS#(0:127)@114)

Address: 4328

```
property stream_scanlist_address115: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS115 (STREAM_SCANLIST_ADDRESS#(0:127)@115)

Address: 4330

```
property stream_scanlist_address116: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS116 (STREAM_SCANLIST_ADDRESS#(0:127)@116)

Address: 4332

```
property stream_scanlist_address117: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS117 (STREAM_SCANLIST_ADDRESS#(0:127)@117)

Address: 4334

```
property stream_scanlist_address118: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS118 (STREAM_SCANLIST_ADDRESS#(0:127)@118)

Address: 4336

```
property stream_scanlist_address119: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS119 (STREAM_SCANLIST_ADDRESS#(0:127)@119)

Address: 4338

```
property stream_scanlist_address12: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS12 (STREAM_SCANLIST_ADDRESS#(0:127)@12)

Address: 4124

```
property stream_scanlist_address120: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS120 (STREAM_SCANLIST_ADDRESS#(0:127)@120)

Address: 4340

```
property stream_scanlist_address121: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS121 (STREAM_SCANLIST_ADDRESS#(0:127)@121)

Address: 4342

```
property stream_scanlist_address122: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS122 (STREAM_SCANLIST_ADDRESS#(0:127)@122)

Address: 4344

```
property stream_scanlist_address123: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS123 (STREAM_SCANLIST_ADDRESS#(0:127)@123)

Address: 4346

```
property stream_scanlist_address124: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS124 (STREAM_SCANLIST_ADDRESS#(0:127)@124)

Address: 4348

```
property stream_scanlist_address125: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS125 (STREAM_SCANLIST_ADDRESS#(0:127)@125)

Address: 4350

```
property stream_scanlist_address126: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS126 (STREAM_SCANLIST_ADDRESS#(0:127)@126)

Address: 4352

```
property stream_scanlist_address127: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS127 (STREAM_SCANLIST_ADDRESS#(0:127)@127)

Address: 4354

```
property stream_scanlist_address13: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS13 (STREAM_SCANLIST_ADDRESS#(0:127)@13)

Address: 4126

```
property stream_scanlist_address14: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS14 (STREAM_SCANLIST_ADDRESS#(0:127)@14)

Address: 4128

```
property stream_scanlist_address15: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS15 (STREAM_SCANLIST_ADDRESS#(0:127)@15)

Address: 4130

```
property stream_scanlist_address16: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS16 (STREAM_SCANLIST_ADDRESS#(0:127)@16)

Address: 4132

```
property stream_scanlist_address17: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS17 (STREAM_SCANLIST_ADDRESS#(0:127)@17)

Address: 4134

```
property stream_scanlist_address18: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS18 (STREAM_SCANLIST_ADDRESS#(0:127)@18)

Address: 4136

```
property stream_scanlist_address19: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS19 (STREAM_SCANLIST_ADDRESS#(0:127)@19)

Address: 4138

```
property stream_scanlist_address2: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS2 (STREAM_SCANLIST_ADDRESS#(0:127)@2)

Address: 4104

```
property stream_scanlist_address20: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS20 (STREAM_SCANLIST_ADDRESS#(0:127)@20)

Address: 4140

```
property stream_scanlist_address21: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS21 (STREAM_SCANLIST_ADDRESS#(0:127)@21)

Address: 4142

```
property stream_scanlist_address22: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS22 (STREAM_SCANLIST_ADDRESS#(0:127)@22)

Address: 4144

```
property stream_scanlist_address23: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS23 (STREAM_SCANLIST_ADDRESS#(0:127)@23)

Address: 4146

```
property stream_scanlist_address24: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS24 (STREAM_SCANLIST_ADDRESS#(0:127)@24)

Address: 4148

```
property stream_scanlist_address25: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS25 (STREAM_SCANLIST_ADDRESS#(0:127)@25)

Address: 4150

```
property stream_scanlist_address26: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS26 (STREAM_SCANLIST_ADDRESS#(0:127)@26)

Address: 4152

```
property stream_scanlist_address27: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS27 (STREAM_SCANLIST_ADDRESS#(0:127)@27)

Address: 4154

```
property stream_scanlist_address28: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS28 (STREAM_SCANLIST_ADDRESS#(0:127)@28)

Address: 4156

```
property stream_scanlist_address29: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS29 (STREAM_SCANLIST_ADDRESS#(0:127)@29)

Address: 4158

```
property stream_scanlist_address3: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS3 (STREAM_SCANLIST_ADDRESS#(0:127)@3)

Address: 4106

```
property stream_scanlist_address30: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS30 (STREAM_SCANLIST_ADDRESS#(0:127)@30)

Address: 4160

```
property stream_scanlist_address31: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS31 (STREAM_SCANLIST_ADDRESS#(0:127)@31)

Address: 4162

```
property stream_scanlist_address32: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS32 (STREAM_SCANLIST_ADDRESS#(0:127)@32)

Address: 4164

```
property stream_scanlist_address33: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS33 (STREAM_SCANLIST_ADDRESS#(0:127)@33)

Address: 4166

```
property stream_scanlist_address34: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS34 (STREAM_SCANLIST_ADDRESS#(0:127)@34)

Address: 4168

```
property stream_scanlist_address35: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS35 (STREAM_SCANLIST_ADDRESS#(0:127)@35)

Address: 4170

```
property stream_scanlist_address36: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS36 (STREAM_SCANLIST_ADDRESS#(0:127)@36)

Address: 4172

```
property stream_scanlist_address37: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS37 (STREAM_SCANLIST_ADDRESS#(0:127)@37)

Address: 4174

```
property stream_scanlist_address38: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS38 (STREAM_SCANLIST_ADDRESS#(0:127)@38)

Address: 4176

```
property stream_scanlist_address39: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS39 (STREAM_SCANLIST_ADDRESS#(0:127)@39)

Address: 4178

```
property stream_scanlist_address4: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS4 (STREAM_SCANLIST_ADDRESS#(0:127)@4)

Address: 4108

```
property stream_scanlist_address40: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS40 (STREAM_SCANLIST_ADDRESS#(0:127)@40)

Address: 4180

```
property stream_scanlist_address41: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS41 (STREAM_SCANLIST_ADDRESS#(0:127)@41)

Address: 4182

```
property stream_scanlist_address42: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS42 (STREAM_SCANLIST_ADDRESS#(0:127)@42)

Address: 4184

```
property stream_scanlist_address43: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS43 (STREAM_SCANLIST_ADDRESS#(0:127)@43)

Address: 4186

```
property stream_scanlist_address44: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS44 (STREAM_SCANLIST_ADDRESS#(0:127)@44)

Address: 4188

```
property stream_scanlist_address45: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS45 (STREAM_SCANLIST_ADDRESS#(0:127)@45)

Address: 4190

```
property stream_scanlist_address46: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS46 (STREAM_SCANLIST_ADDRESS#(0:127)@46)

Address: 4192

```
property stream_scanlist_address47: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS47 (STREAM_SCANLIST_ADDRESS#(0:127)@47)

Address: 4194

```
property stream_scanlist_address48: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS48 (STREAM_SCANLIST_ADDRESS#(0:127)@48)

Address: 4196

```
property stream_scanlist_address49: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS49 (STREAM_SCANLIST_ADDRESS#(0:127)@49)

Address: 4198

```
property stream_scanlist_address5: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS5 (STREAM_SCANLIST_ADDRESS#(0:127)@5)

Address: 4110

```
property stream_scanlist_address50: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS50 (STREAM_SCANLIST_ADDRESS#(0:127)@50)

Address: 4200

```
property stream_scanlist_address51: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS51 (STREAM_SCANLIST_ADDRESS#(0:127)@51)

Address: 4202

```
property stream_scanlist_address52: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS52 (STREAM_SCANLIST_ADDRESS#(0:127)@52)

Address: 4204

```
property stream_scanlist_address53: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS53 (STREAM_SCANLIST_ADDRESS#(0:127)@53)

Address: 4206

```
property stream_scanlist_address54: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS54 (STREAM_SCANLIST_ADDRESS#(0:127)@54)

Address: 4208

```
property stream_scanlist_address55: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS55 (STREAM_SCANLIST_ADDRESS#(0:127)@55)

Address: 4210

```
property stream_scanlist_address56: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS56 (STREAM_SCANLIST_ADDRESS#(0:127)@56)

Address: 4212

```
property stream_scanlist_address57: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS57 (STREAM_SCANLIST_ADDRESS#(0:127)@57)

Address: 4214

```
property stream_scanlist_address58: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS58 (STREAM_SCANLIST_ADDRESS#(0:127)@58)

Address: 4216

```
property stream_scanlist_address59: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS59 (STREAM_SCANLIST_ADDRESS#(0:127)@59)

Address: 4218

```
property stream_scanlist_address6: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS6 (STREAM_SCANLIST_ADDRESS#(0:127)@6)

Address: 4112

```
property stream_scanlist_address60: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS60 (STREAM_SCANLIST_ADDRESS#(0:127)@60)

Address: 4220

```
property stream_scanlist_address61: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS61 (STREAM_SCANLIST_ADDRESS#(0:127)@61)

Address: 4222

```
property stream_scanlist_address62: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS62 (STREAM_SCANLIST_ADDRESS#(0:127)@62)

Address: 4224

```
property stream_scanlist_address63: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS63 (STREAM_SCANLIST_ADDRESS#(0:127)@63)

Address: 4226

```
property stream_scanlist_address64: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS64 (STREAM_SCANLIST_ADDRESS#(0:127)@64)

Address: 4228

```
property stream_scanlist_address65: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS65 (STREAM_SCANLIST_ADDRESS#(0:127)@65)

Address: 4230

```
property stream_scanlist_address66: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS66 (STREAM_SCANLIST_ADDRESS#(0:127)@66)

Address: 4232

```
property stream_scanlist_address67: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS67 (STREAM_SCANLIST_ADDRESS#(0:127)@67)

Address: 4234

```
property stream_scanlist_address68: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS68 (STREAM_SCANLIST_ADDRESS#(0:127)@68)

Address: 4236

```
property stream_scanlist_address69: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS69 (STREAM_SCANLIST_ADDRESS#(0:127)@69)

Address: 4238

```
property stream_scanlist_address7: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS7 (STREAM_SCANLIST_ADDRESS#(0:127)@7)

Address: 4114

```
property stream_scanlist_address70: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS70 (STREAM_SCANLIST_ADDRESS#(0:127)@70)

Address: 4240

```
property stream_scanlist_address71: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS71 (STREAM_SCANLIST_ADDRESS#(0:127)@71)

Address: 4242

```
property stream_scanlist_address72: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS72 (STREAM_SCANLIST_ADDRESS#(0:127)@72)

Address: 4244

```
property stream_scanlist_address73: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS73 (STREAM_SCANLIST_ADDRESS#(0:127)@73)

Address: 4246

```
property stream_scanlist_address74: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS74 (STREAM_SCANLIST_ADDRESS#(0:127)@74)

Address: 4248

```
property stream_scanlist_address75: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS75 (STREAM_SCANLIST_ADDRESS#(0:127)@75)

Address: 4250

```
property stream_scanlist_address76: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS76 (STREAM_SCANLIST_ADDRESS#(0:127)@76)

Address: 4252

```
property stream_scanlist_address77: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS77 (STREAM_SCANLIST_ADDRESS#(0:127)@77)

Address: 4254

```
property stream_scanlist_address78: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS78 (STREAM_SCANLIST_ADDRESS#(0:127)@78)

Address: 4256

```
property stream_scanlist_address79: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS79 (STREAM_SCANLIST_ADDRESS#(0:127)@79)

Address: 4258

```
property stream_scanlist_address8: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS8 (STREAM_SCANLIST_ADDRESS#(0:127)@8)

Address: 4116

```
property stream_scanlist_address80: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS80 (STREAM_SCANLIST_ADDRESS#(0:127)@80)

Address: 4260

```
property stream_scanlist_address81: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS81 (STREAM_SCANLIST_ADDRESS#(0:127)@81)

Address: 4262

```
property stream_scanlist_address82: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS82 (STREAM_SCANLIST_ADDRESS#(0:127)@82)

Address: 4264

```
property stream_scanlist_address83: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS83 (STREAM_SCANLIST_ADDRESS#(0:127)@83)

Address: 4266

```
property stream_scanlist_address84: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS84 (STREAM_SCANLIST_ADDRESS#(0:127)@84)

Address: 4268

```
property stream_scanlist_address85: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS85 (STREAM_SCANLIST_ADDRESS#(0:127)@85)

Address: 4270

```
property stream_scanlist_address86: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS86 (STREAM_SCANLIST_ADDRESS#(0:127)@86)

Address: 4272

```
property stream_scanlist_address87: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS87 (STREAM_SCANLIST_ADDRESS#(0:127)@87)

Address: 4274

```
property stream_scanlist_address88: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS88 (STREAM_SCANLIST_ADDRESS#(0:127)@88)

Address: 4276

```
property stream_scanlist_address89: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS89 (STREAM_SCANLIST_ADDRESS#(0:127)@89)

Address: 4278

```
property stream_scanlist_address9: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS9 (STREAM_SCANLIST_ADDRESS#(0:127)@9)

Address: 4118

```
property stream_scanlist_address90: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS90 (STREAM_SCANLIST_ADDRESS#(0:127)@90)

Address: 4280

```
property stream_scanlist_address91: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS91 (STREAM_SCANLIST_ADDRESS#(0:127)@91)

Address: 4282

```
property stream_scanlist_address92: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS92 (STREAM_SCANLIST_ADDRESS#(0:127)@92)

Address: 4284

```
property stream_scanlist_address93: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS93 (STREAM_SCANLIST_ADDRESS#(0:127)@93)

Address: 4286

```
property stream_scanlist_address94: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS94 (STREAM_SCANLIST_ADDRESS#(0:127)@94)

Address: 4288

```
property stream_scanlist_address95: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS95 (STREAM_SCANLIST_ADDRESS#(0:127)@95)

Address: 4290

```
property stream_scanlist_address96: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS96 (STREAM_SCANLIST_ADDRESS#(0:127)@96)

Address: 4292

```
property stream_scanlist_address97: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS97 (STREAM_SCANLIST_ADDRESS#(0:127)@97)

Address: 4294

```
property stream_scanlist_address98: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS98 (STREAM_SCANLIST_ADDRESS#(0:127)@98)

Address: 4296

```
property stream_scanlist_address99: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS99 (STREAM_SCANLIST_ADDRESS#(0:127)@99)

Address: 4298

```
property stream_scanrate_hz: float
```

Write a value to specify the number of times per second that all channels in the stream scanlist will be read. Max stream speeds are based on Sample Rate which is NumChannels*ScanRate. Has no effect when using an external clock. A read of this register returns the actual scan rate, which can be slightly different due to rounding. For scan rates >152.588, the actual scan interval is multiples of 100 ns. Assuming a core clock of 80 MHz the internal roll value is (80M/(8*DesiredScanRate))-1 and the actual scan rate is then 80M/(8*(RollValue+1). For slower scan rates the scan interval resolution is changed to 1 us, 10 us, 100 us, or 1 ms as needed to achieve the longer intervals.

Register: STREAM_SCANRATE_HZ

Address: 4002

```
property stream_settling_us: float
```

Time in microseconds to allow signals to settle after switching the mux. Does not apply to the 1st channel in the scan list, as that settling is controlled by scan rate (the time from the last channel until the start of the next scan). Default=0. When set to less than 1, automatic settling will be used. The automatic settling behavior varies by device.

Register: STREAM_SETTLING_US

Address: 4008

```
property stream_start_time_stamp: int
```

This register stores the value of CORE_TIMER at the start of the first stream scan. Note that the first stream scan happens 1 scan period after STREAM_ENABLE is set to 1.

Register: STREAM_START_TIME_STAMP

Address: 4026

```
property stream_trigger_index: int
```

Controls when stream scanning will start. 0 = Start when stream is enabled, 2000 = Start when DIO_EF0 detects an edge, 2001 = Start when DIO_EF1 detects an edge. See the stream documentation for all supported values.

Register: STREAM_TRIGGER_INDEX

Address: 4024

```
property system_counter_10khz: int
```

A 10 kHz counter synchronized to RTC_TIME_S. This register can be appended to RTC_TIME_S as the decimal portion to get 0.1 ms resolution.

Register: SYSTEM_COUNTER_10KHZ

Address: 61502

```
property system_reboot: int
```

Issues a device reboot. Must write 0x4C4Axxxx, where xxxx is number of 50ms ticks to wait before reboot. To reboot immediately write 0x4C4A0000 (d1279918080).

Register: SYSTEM_REBOOT

Address: 61998

```
property system_timer_20hz: int
```

Internal 32-bit system timer running at 20Hz.

Register: SYSTEM_TIMER_20HZ

Address: 61522

```
property tdac0: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC0 (TDAC#(0:21)@0)

Address: 30000

```
property tdac1: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC1 (TDAC#(0:21)@1)

Address: 30002

```
property tdac10: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC10 (TDAC#(0:21)@10)

Address: 30020

```
property tdac11: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC11 (TDAC#(0:21)@11)

Address: 30022

```
property tdac12: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC12 (TDAC#(0:21)@12)

Address: 30024

```
property tdac13: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC13 (TDAC#(0:21)@13)

Address: 30026

```
property tdac14: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC14 (TDAC#(0:21)@14)

Address: 30028

```
property tdac15: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC15 (TDAC#(0:21)@15)

Address: 30030

```
property tdac16: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC16 (TDAC#(0:21)@16)

Address: 30032

```
property tdac17: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC17 (TDAC#(0:21)@17)

Address: 30034

```
property tdac18: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC18 (TDAC#(0:21)@18)

Address: 30036

```
property tdac19: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC19 (TDAC#(0:21)@19)

Address: 30038

```
property tdac2: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC2 (TDAC#(0:21)@2)

Address: 30004

```
property tdac20: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC20 (TDAC#(0:21)@20)

Address: 30040

```
property tdac21: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC21 (TDAC#(0:21)@21)

Address: 30042

```
property tdac3: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC3 (TDAC#(0:21)@3)

Address: 30006

```
property tdac4: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC4 (TDAC#(0:21)@4)

Address: 30008

```
property tdac5: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC5 (TDAC#(0:21)@5)

Address: 30010

```
property tdac6: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC6 (TDAC#(0:21)@6)

Address: 30012

```
property tdac7: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC7 (TDAC#(0:21)@7)

Address: 30014

```
property tdac8: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC8 (TDAC#(0:21)@8)

Address: 30016

```
property tdac9: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC9 (TDAC#(0:21)@9)

Address: 30018

```
property tdac_serial_number: int
```

Returns the serial number of an LJTick-DAC, and forces a re-read of the calibration constants. Which LJTDAC is determined by the last write to TDAC# ... whether it was successful or not.

Register: TDAC_SERIAL_NUMBER

Address: 55200

```
property tdac_speed_throttle: int
```

Sets the I2C clock speed that will be used when communicating with the TDAC. Default value is 65516. See I2C_SPEED_THROTTLE for more detail.

Register: TDAC_SPEED_THROTTLE

Address: 55202

```
property temperature_air_k: float
```

Returns the estimated ambient air temperature just outside of the device in its red plastic enclosure. This register is equal to TEMPERATURE_DEVICE_K - 4.3. If Ethernet and/or WiFi is enabled, subtract an extra 0.6 for each.

Register: TEMPERATURE_AIR_K

Address: 60050

```
property temperature_device_k: float
```

Takes a reading from the internal temperature sensor using range=+/-10V and resolution=8, and applies the formula Volts*-92.6+467.6 to return kelvins.

Register: TEMPERATURE_DEVICE_K

Address: 60052

```
property test: int
```

A read of this test register should always return 0x00112233 or d1122867. If your software has the word swap quirk, you will incorrectly read 0x22330011 or 573767697. If your software has the address-1 quirk, a UINT16 (1-register) read from 55101 will incorrectly return 0x0011 (should read 0x2233). This register is unlike others, in that it allows you can read a single word from 55100 or 55101, or of course 2 words from 55100.

Register: TEST

Address: 55100

```
property test_float32: float
```

Write a value and read back to test FLOAT32 operation. Default is 0xC61C3C00 or -9999.0. If your software has the word swap quirk, the default will incorrectly read 0x3C00C61C or 0.00786.

Register: TEST_FLOAT32

Address: 55124

```
property test_int32: int
```

Write a value and read back to test INT32 operation. Default is 0x8899AABB or d-2003195205. If your software has the word swap quirk, the default will incorrectly read 0xAABB8899 or -1430550375.

Register: TEST_INT32

Address: 55122

```
property test_uint16: int
```

Write a value and read back to test UINT16 operation. Default is 0x0011 or d17.

Register: TEST_UINT16

Address: 55110

```
property test_uint32: int
```

Write a value and read back to test UINT32 operation. Default is 0x00112233 or d1122867. If your software has the word swap quirk, the default will incorrectly read 0x22330011 or 573767697.

Register: TEST_UINT32

Address: 55120

```
property user_ram0_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM0_F32 (USER_RAM#(0:39)_F32@0)

Address: 46000

```
property user_ram0_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM0_I32 (USER_RAM#(0:9)_I32@0)

Address: 46080

```
property user_ram0_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM0_U16 (USER_RAM#(0:19)_U16@0)

Address: 46180

```
property user_ram0_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM0_U32 (USER_RAM#(0:39)_U32@0)

Address: 46100

```
property user_ram10_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM10_F32 (USER_RAM#(0:39)_F32@10)

Address: 46020

```
property user_ram10_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM10_U16 (USER_RAM#(0:19)_U16@10)

Address: 46190

```
property user_ram10_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM10_U32 (USER_RAM#(0:39)_U32@10)

Address: 46120

```
property user_ram11_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM11_F32 (USER_RAM#(0:39)_F32@11)

Address: 46022

```
property user_ram11_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM11_U16 (USER_RAM#(0:19)_U16@11)

Address: 46191

```
property user_ram11_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM11_U32 (USER_RAM#(0:39)_U32@11)

Address: 46122

```
property user_ram12_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM12_F32 (USER_RAM#(0:39)_F32@12)

Address: 46024

```
property user_ram12_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM12_U16 (USER_RAM#(0:19)_U16@12)

Address: 46192

```
property user_ram12_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM12_U32 (USER_RAM#(0:39)_U32@12)

Address: 46124

```
property user_ram13_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM13_F32 (USER_RAM#(0:39)_F32@13)

Address: 46026

```
property user_ram13_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM13_U16 (USER_RAM#(0:19)_U16@13)

Address: 46193

```
property user_ram13_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM13_U32 (USER_RAM#(0:39)_U32@13)

Address: 46126

```
property user_ram14_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM14_F32 (USER_RAM#(0:39)_F32@14)

Address: 46028

```
property user_ram14_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM14_U16 (USER_RAM#(0:19)_U16@14)

Address: 46194

```
property user_ram14_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM14_U32 (USER_RAM#(0:39)_U32@14)

Address: 46128

```
property user_ram15_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM15_F32 (USER_RAM#(0:39)_F32@15)

Address: 46030

```
property user_ram15_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM15_U16 (USER_RAM#(0:19)_U16@15)

Address: 46195

```
property user_ram15_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM15_U32 (USER_RAM#(0:39)_U32@15)

Address: 46130

```
property user_ram16_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM16_F32 (USER_RAM#(0:39)_F32@16)

Address: 46032

```
property user_ram16_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM16_U16 (USER_RAM#(0:19)_U16@16)

Address: 46196

```
property user_ram16_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM16_U32 (USER_RAM#(0:39)_U32@16)

Address: 46132

```
property user_ram17_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM17_F32 (USER_RAM#(0:39)_F32@17)

Address: 46034

```
property user_ram17_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM17_U16 (USER_RAM#(0:19)_U16@17)

Address: 46197

```
property user_ram17_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM17_U32 (USER_RAM#(0:39)_U32@17)

Address: 46134

```
property user_ram18_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM18_F32 (USER_RAM#(0:39)_F32@18)

Address: 46036

```
property user_ram18_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM18_U16 (USER_RAM#(0:19)_U16@18)

Address: 46198

```
property user_ram18_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM18_U32 (USER_RAM#(0:39)_U32@18)

Address: 46136

```
property user_ram19_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM19_F32 (USER_RAM#(0:39)_F32@19)

Address: 46038

```
property user_ram19_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM19_U16 (USER_RAM#(0:19)_U16@19)

Address: 46199

```
property user_ram19_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM19_U32 (USER_RAM#(0:39)_U32@19)

Address: 46138

```
property user_ram1_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM1_F32 (USER_RAM#(0:39)_F32@1)

Address: 46002

```
property user_ram1_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM1_I32 (USER_RAM#(0:9)_I32@1)

Address: 46082

```
property user_ram1_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM1_U16 (USER_RAM#(0:19)_U16@1)

Address: 46181

```
property user_ram1_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM1_U32 (USER_RAM#(0:39)_U32@1)

Address: 46102

```
property user_ram20_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM20_F32 (USER_RAM#(0:39)_F32@20)

Address: 46040

```
property user_ram20_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM20_U32 (USER_RAM#(0:39)_U32@20)

Address: 46140

```
property user_ram21_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM21_F32 (USER_RAM#(0:39)_F32@21)

Address: 46042

```
property user_ram21_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM21_U32 (USER_RAM#(0:39)_U32@21)

Address: 46142

```
property user_ram22_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM22_F32 (USER_RAM#(0:39)_F32@22)

Address: 46044

```
property user_ram22_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM22_U32 (USER_RAM#(0:39)_U32@22)

Address: 46144

```
property user_ram23_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM23_F32 (USER_RAM#(0:39)_F32@23)

Address: 46046

```
property user_ram23_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM23_U32 (USER_RAM#(0:39)_U32@23)

Address: 46146

```
property user_ram24_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM24_F32 (USER_RAM#(0:39)_F32@24)

Address: 46048

```
property user_ram24_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM24_U32 (USER_RAM#(0:39)_U32@24)

Address: 46148

```
property user_ram25_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM25_F32 (USER_RAM#(0:39)_F32@25)

Address: 46050

```
property user_ram25_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM25_U32 (USER_RAM#(0:39)_U32@25)

Address: 46150

```
property user_ram26_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM26_F32 (USER_RAM#(0:39)_F32@26)

Address: 46052

```
property user_ram26_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM26_U32 (USER_RAM#(0:39)_U32@26)

Address: 46152

```
property user_ram27_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM27_F32 (USER_RAM#(0:39)_F32@27)

Address: 46054

```
property user_ram27_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM27_U32 (USER_RAM#(0:39)_U32@27)

Address: 46154

```
property user_ram28_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM28_F32 (USER_RAM#(0:39)_F32@28)

Address: 46056

```
property user_ram28_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM28_U32 (USER_RAM#(0:39)_U32@28)

Address: 46156

```
property user_ram29_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM29_F32 (USER_RAM#(0:39)_F32@29)

Address: 46058

```
property user_ram29_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM29_U32 (USER_RAM#(0:39)_U32@29)

Address: 46158

```
property user_ram2_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM2_F32 (USER_RAM#(0:39)_F32@2)

Address: 46004

```
property user_ram2_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM2_I32 (USER_RAM#(0:9)_I32@2)

Address: 46084

```
property user_ram2_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM2_U16 (USER_RAM#(0:19)_U16@2)

Address: 46182

```
property user_ram2_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM2_U32 (USER_RAM#(0:39)_U32@2)

Address: 46104

```
property user_ram30_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM30_F32 (USER_RAM#(0:39)_F32@30)

Address: 46060

```
property user_ram30_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM30_U32 (USER_RAM#(0:39)_U32@30)

Address: 46160

```
property user_ram31_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM31_F32 (USER_RAM#(0:39)_F32@31)

Address: 46062

```
property user_ram31_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM31_U32 (USER_RAM#(0:39)_U32@31)

Address: 46162

```
property user_ram32_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM32_F32 (USER_RAM#(0:39)_F32@32)

Address: 46064

```
property user_ram32_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM32_U32 (USER_RAM#(0:39)_U32@32)

Address: 46164

```
property user_ram33_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM33_F32 (USER_RAM#(0:39)_F32@33)

Address: 46066

```
property user_ram33_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM33_U32 (USER_RAM#(0:39)_U32@33)

Address: 46166

```
property user_ram34_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM34_F32 (USER_RAM#(0:39)_F32@34)

Address: 46068

```
property user_ram34_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM34_U32 (USER_RAM#(0:39)_U32@34)

Address: 46168

```
property user_ram35_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM35_F32 (USER_RAM#(0:39)_F32@35)

Address: 46070

```
property user_ram35_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM35_U32 (USER_RAM#(0:39)_U32@35)

Address: 46170

```
property user_ram36_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM36_F32 (USER_RAM#(0:39)_F32@36)

Address: 46072

```
property user_ram36_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM36_U32 (USER_RAM#(0:39)_U32@36)

Address: 46172

```
property user_ram37_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM37_F32 (USER_RAM#(0:39)_F32@37)

Address: 46074

```
property user_ram37_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM37_U32 (USER_RAM#(0:39)_U32@37)

Address: 46174

```
property user_ram38_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM38_F32 (USER_RAM#(0:39)_F32@38)

Address: 46076

```
property user_ram38_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM38_U32 (USER_RAM#(0:39)_U32@38)

Address: 46176

```
property user_ram39_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM39_F32 (USER_RAM#(0:39)_F32@39)

Address: 46078

```
property user_ram39_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM39_U32 (USER_RAM#(0:39)_U32@39)

Address: 46178

```
property user_ram3_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM3_F32 (USER_RAM#(0:39)_F32@3)

Address: 46006

```
property user_ram3_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM3_I32 (USER_RAM#(0:9)_I32@3)

Address: 46086

```
property user_ram3_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM3_U16 (USER_RAM#(0:19)_U16@3)

Address: 46183

```
property user_ram3_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM3_U32 (USER_RAM#(0:39)_U32@3)

Address: 46106

```
property user_ram4_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM4_F32 (USER_RAM#(0:39)_F32@4)

Address: 46008

```
property user_ram4_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM4_I32 (USER_RAM#(0:9)_I32@4)

Address: 46088

```
property user_ram4_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM4_U16 (USER_RAM#(0:19)_U16@4)

Address: 46184

```
property user_ram4_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM4_U32 (USER_RAM#(0:39)_U32@4)

Address: 46108

```
property user_ram5_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM5_F32 (USER_RAM#(0:39)_F32@5)

Address: 46010

```
property user_ram5_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM5_I32 (USER_RAM#(0:9)_I32@5)

Address: 46090

```
property user_ram5_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM5_U16 (USER_RAM#(0:19)_U16@5)

Address: 46185

```
property user_ram5_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM5_U32 (USER_RAM#(0:39)_U32@5)

Address: 46110

```
property user_ram6_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM6_F32 (USER_RAM#(0:39)_F32@6)

Address: 46012

```
property user_ram6_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM6_I32 (USER_RAM#(0:9)_I32@6)

Address: 46092

```
property user_ram6_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM6_U16 (USER_RAM#(0:19)_U16@6)

Address: 46186

```
property user_ram6_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM6_U32 (USER_RAM#(0:39)_U32@6)

Address: 46112

```
property user_ram7_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM7_F32 (USER_RAM#(0:39)_F32@7)

Address: 46014

```
property user_ram7_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM7_I32 (USER_RAM#(0:9)_I32@7)

Address: 46094

```
property user_ram7_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM7_U16 (USER_RAM#(0:19)_U16@7)

Address: 46187

```
property user_ram7_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM7_U32 (USER_RAM#(0:39)_U32@7)

Address: 46114

```
property user_ram8_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM8_F32 (USER_RAM#(0:39)_F32@8)

Address: 46016

```
property user_ram8_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM8_I32 (USER_RAM#(0:9)_I32@8)

Address: 46096

```
property user_ram8_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM8_U16 (USER_RAM#(0:19)_U16@8)

Address: 46188

```
property user_ram8_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM8_U32 (USER_RAM#(0:39)_U32@8)

Address: 46116

```
property user_ram9_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM9_F32 (USER_RAM#(0:39)_F32@9)

Address: 46018

```
property user_ram9_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM9_I32 (USER_RAM#(0:9)_I32@9)

Address: 46098

```
property user_ram9_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM9_U16 (USER_RAM#(0:19)_U16@9)

Address: 46189

```
property user_ram9_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM9_U32 (USER_RAM#(0:39)_U32@9)

Address: 46118

```
property user_ram_fifo0_allocate_num_bytes: int
```

Allocate memory for a FIFO buffer. Number of bytes should be sufficient to store users max transfer array size. Note that FLOAT32, INT32, and UINT32 require 4 bytes per value, and UINT16 require 2 bytes per value. Maximum size is limited by available memory. Care should be taken to conserve enough memory for other operations such as AIN_EF, Lua, Stream etc.

Register: USER_RAM_FIFO0_ALLOCATE_NUM_BYTES
(USER_RAM_FIFO#(0:3)_ALLOCATE_NUM_BYTES@0)

Address: 47900

```
property user_ram_fifo0_data_f32: float
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO0_DATA_F32 (USER_RAM_FIFO#(0:3)_DATA_F32@0)

Address: 47030

```
property user_ram_fifo0_data_i32: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO0_DATA_I32 (USER_RAM_FIFO#(0:3)_DATA_I32@0)

Address: 47020

```
property user_ram_fifo0_data_u16: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO0_DATA_U16 (USER_RAM_FIFO#(0:3)_DATA_U16@0)

Address: 47000

```
property user_ram_fifo0_data_u32: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO0_DATA_U32 (USER_RAM_FIFO#(0:3)_DATA_U32@0)

Address: 47010

```
property user_ram_fifo0_empty: int
```

Write any value to this register to efficiently empty, flush, or otherwise clear data from the FIFO.

Register: USER_RAM_FIFO0_EMPTY (USER_RAM_FIFO#(0:3)_EMPTY@0)

Address: 47930

```
property user_ram_fifo0_num_bytes_in_fifo: int
```

Poll this register to see when new data is available/ready. Each read of the FIFO buffer decreases this value, and each write to the FIFO buffer increases this value.

At any point in time, the following equation holds: Nbytes = Nwritten - Nread.

Register: USER_RAM_FIFO0_NUM_BYTES_IN_FIFO
(USER_RAM_FIFO#(0:3)_NUM_BYTES_IN_FIFO@0)

Address: 47910

```
property user_ram_fifo1_allocate_num_bytes: int
```

Allocate memory for a FIFO buffer. Number of bytes should be sufficient to store users max transfer array size. Note that FLOAT32, INT32, and UINT32 require 4 bytes per value, and UINT16 require 2 bytes per value. Maximum size is limited by available memory. Care should be taken to conserve enough memory for other operations such as AIN_EF, Lua, Stream etc.

Register: USER_RAM_FIFO1_ALLOCATE_NUM_BYTES
(USER_RAM_FIFO#(0:3)_ALLOCATE_NUM_BYTES@1)

Address: 47902

```
property user_ram_fifo1_data_f32: float
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO1_DATA_F32 (USER_RAM_FIFO#(0:3)_DATA_F32@1)

Address: 47032

```
property user_ram_fifo1_data_i32: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO1_DATA_I32 (USER_RAM_FIFO#(0:3)_DATA_I32@1)

Address: 47022

```
property user_ram_fifo1_data_u16: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO1_DATA_U16 (USER_RAM_FIFO#(0:3)_DATA_U16@1)

Address: 47001

```
property user_ram_fifo1_data_u32: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO1_DATA_U32 (USER_RAM_FIFO#(0:3)_DATA_U32@1)

Address: 47012

```
property user_ram_fifo1_empty: int
```

Write any value to this register to efficiently empty, flush, or otherwise clear data from the FIFO.

Register: USER_RAM_FIFO1_EMPTY (USER_RAM_FIFO#(0:3)_EMPTY@1)

Address: 47932

```
property user_ram_fifo1_num_bytes_in_fifo: int
```

Poll this register to see when new data is available/ready. Each read of the FIFO buffer decreases this value, and each write to the FIFO buffer increases this value.

At any point in time, the following equation holds: Nbytes = Nwritten - Nread.

Register: USER_RAM_FIFO1_NUM_BYTES_IN_FIFO
(USER_RAM_FIFO#(0:3)_NUM_BYTES_IN_FIFO@1)

Address: 47912

```
property user_ram_fifo2_allocate_num_bytes: int
```

Allocate memory for a FIFO buffer. Number of bytes should be sufficient to store users max transfer array size. Note that FLOAT32, INT32, and UINT32 require 4 bytes per value, and UINT16 require 2 bytes per value. Maximum size is limited by available memory. Care should be taken to conserve enough memory for other operations such as AIN_EF, Lua, Stream etc.

Register: USER_RAM_FIFO2_ALLOCATE_NUM_BYTES
(USER_RAM_FIFO#(0:3)_ALLOCATE_NUM_BYTES@2)

Address: 47904

```
property user_ram_fifo2_data_f32: float
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO2_DATA_F32 (USER_RAM_FIFO#(0:3)_DATA_F32@2)

Address: 47034

```
property user_ram_fifo2_data_i32: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO2_DATA_I32 (USER_RAM_FIFO#(0:3)_DATA_I32@2)

Address: 47024

```
property user_ram_fifo2_data_u16: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO2_DATA_U16 (USER_RAM_FIFO#(0:3)_DATA_U16@2)

Address: 47002

```
property user_ram_fifo2_data_u32: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO2_DATA_U32 (USER_RAM_FIFO#(0:3)_DATA_U32@2)

Address: 47014

```
property user_ram_fifo2_empty: int
```

Write any value to this register to efficiently empty, flush, or otherwise clear data from the FIFO.

Register: USER_RAM_FIFO2_EMPTY (USER_RAM_FIFO#(0:3)_EMPTY@2)

Address: 47934

```
property user_ram_fifo2_num_bytes_in_fifo: int
```

Poll this register to see when new data is available/ready. Each read of the FIFO buffer decreases this value, and each write to the FIFO buffer increases this value.

At any point in time, the following equation holds: Nbytes = Nwritten - Nread.

Register: USER_RAM_FIFO2_NUM_BYTES_IN_FIFO
(USER_RAM_FIFO#(0:3)_NUM_BYTES_IN_FIFO@2)

Address: 47914

```
property user_ram_fifo3_allocate_num_bytes: int
```

Allocate memory for a FIFO buffer. Number of bytes should be sufficient to store users max transfer array size. Note that FLOAT32, INT32, and UINT32 require 4 bytes per value, and UINT16 require 2 bytes per value. Maximum size is limited by available memory. Care should be taken to conserve enough memory for other operations such as AIN_EF, Lua, Stream etc.

Register: USER_RAM_FIFO3_ALLOCATE_NUM_BYTES
(USER_RAM_FIFO#(0:3)_ALLOCATE_NUM_BYTES@3)

Address: 47906

```
property user_ram_fifo3_data_f32: float
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO3_DATA_F32 (USER_RAM_FIFO#(0:3)_DATA_F32@3)

Address: 47036

```
property user_ram_fifo3_data_i32: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO3_DATA_I32 (USER_RAM_FIFO#(0:3)_DATA_I32@3)

Address: 47026

```
property user_ram_fifo3_data_u16: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO3_DATA_U16 (USER_RAM_FIFO#(0:3)_DATA_U16@3)

Address: 47003

```
property user_ram_fifo3_data_u32: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO3_DATA_U32 (USER_RAM_FIFO#(0:3)_DATA_U32@3)

Address: 47016

```
property user_ram_fifo3_empty: int
```

Write any value to this register to efficiently empty, flush, or otherwise clear data from the FIFO.

Register: USER_RAM_FIFO3_EMPTY (USER_RAM_FIFO#(0:3)_EMPTY@3)

Address: 47936

```
property user_ram_fifo3_num_bytes_in_fifo: int
```

Poll this register to see when new data is available/ready. Each read of the FIFO buffer decreases this value, and each write to the FIFO buffer increases this value.

At any point in time, the following equation holds: Nbytes = Nwritten - Nread.

Register: USER_RAM_FIFO3_NUM_BYTES_IN_FIFO
(USER_RAM_FIFO#(0:3)_NUM_BYTES_IN_FIFO@3)

Address: 47916

```
property wait_us_blocking: int
```

Delays for x microseconds. Range is 0-100000. This operation is Blocking. While a blocking function is running no other registers can be read / written.

Register: WAIT_US_BLOCKING

Address: 61590

```
property watchdog_advanced_default: int
```

A single binary-encoded value where each bit is an advanced option. If bit 0 is set, IO_CONFIG_SET_CURRENT_TO_FACTORY will be done on timeout. If bit 1 is set, IO_CONFIG_SET_CURRENT_TO_DEFAULT will be done on timeout.

Register: WATCHDOG_ADVANCED_DEFAULT

Address: 61602

```
property watchdog_dac0_default: float
```

The voltage of DAC0 after a Watchdog timeout.

Register: WATCHDOG_DAC0_DEFAULT

Address: 61642

```
property watchdog_dac0_enable_default: int
```

Timeout action: Set to 1 to enable DAC0 update on watchdog timeout.

Register: WATCHDOG_DAC0_ENABLE_DEFAULT

Address: 61640

```
property watchdog_dac1_default: float
```

The voltage of DAC1 after a Watchdog timeout.

Register: WATCHDOG_DAC1_DEFAULT

Address: 61652

```
property watchdog_dac1_enable_default: int
```

Timeout action: Set to 1 to enable DAC1 update on watchdog timeout.

Register: WATCHDOG_DAC1_ENABLE_DEFAULT

Address: 61650

```
property watchdog_dio_direction_default: int
```

The direction input/output of the digital I/O after a Watchdog timeout. See DIO_DIRECTION.

Register: WATCHDOG_DIO_DIRECTION_DEFAULT

Address: 61634

```
property watchdog_dio_enable_default: int
```

Timeout action: Set to 1 to enable DIO update on watchdog timeout.

Register: WATCHDOG_DIO_ENABLE_DEFAULT

Address: 61630

```
property watchdog_dio_inhibit_default: int
```

This register can be used to prevent the watchdog from changing specific IO. See DIO_INHIBIT for more information.

Register: WATCHDOG_DIO_INHIBIT_DEFAULT

Address: 61636

```
property watchdog_dio_state_default: int
```

The state high/low of the digital I/O after a Watchdog timeout. See DIO_STATE.

Register: WATCHDOG_DIO_STATE_DEFAULT

Address: 61632

```
property watchdog_enable_default: int
```

Write a 1 to enable the watchdog or a 0 to disable. The watchdog must be disabled before writing any of the other watchdog registers (except for WATCHDOG_STRICT_CLEAR).

Register: WATCHDOG_ENABLE_DEFAULT

Address: 61600

```
property watchdog_reset_enable_default: int
```

Timeout action: Set to 1 to enable device-reset on watchdog timeout.

Register: WATCHDOG_RESET_ENABLE_DEFAULT

Address: 61620

```
property watchdog_startup_delay_s_default: int
```

This specifies the initial timeout period at device bootup. This is used until the first time the watchdog is cleared or timeout ... after that the normal timeout is used.

Register: WATCHDOG_STARTUP_DELAY_S_DEFAULT

Address: 61606

```
property watchdog_strict_clear: int
```

When running in strict mode, writing the key to this register is the only way to clear the watchdog. Writing to this register while not using strict mode will clear the watchdog.

Register: WATCHDOG_STRICT_CLEAR

Address: 61614

```
property watchdog_strict_enable_default: int
```

Set to 1 to enable strict mode.

Register: WATCHDOG_STRICT_ENABLE_DEFAULT

Address: 61610

```
property watchdog_strict_key_default: int
```

When set to strict mode, this is the value that must be written to the clear register.

Register: WATCHDOG_STRICT_KEY_DEFAULT

Address: 61612

```
property watchdog_timeout_s_default: int
```

When the device receives any communication over USB/Ethernet/WiFi, the watchdog timer is cleared. If the watchdog timer is not cleared within the timeout period, the enabled actions will be done.

Register: WATCHDOG_TIMEOUT_S_DEFAULT

Address: 61604

```
property wifi_apply_settings: int
```

Apply all new WiFi settings: IP, Subnet, Gateway, DHCP, SSID, Password. 1=Apply

Register: WIFI_APPLY_SETTINGS

Address: 49400

```
property wifi_dhcp_enable: int
```

Read the current Enabled/Disabled state of WiFi DHCP.

Register: WIFI_DHCP_ENABLE

Address: 49210

```
property wifi_dhcp_enable_default: int
```

The new Enabled/Disabled state of WiFi DHCP. Use WIFI_APPLY_SETTINGS.

Register: WIFI_DHCP_ENABLE_DEFAULT

Address: 49260

```
property wifi_firmware_update_status: int
```

CONFIGURING = 2920, IN_PROGRESS = 2921, REBOOTING = 2923, UPDATE_SUCCESS = 2924, UPDATE_FAILED = 2925.

Register: WIFI_FIRMWARE_UPDATE_STATUS

Address: 49454

```
property wifi_firmware_update_to_versionx: float
```

Start an update by using USB or Ethernet to write the desired version to update to.

Register: WIFI_FIRMWARE_UPDATE_TO_VERSIONX

Address: 49402

```
property wifi_gateway: str
```

Read the current gateway of WiFi.

Register: WIFI_GATEWAY

Address: 49204

```
property wifi_gateway_default: str
```

The new gateway of WiFi. Use WIFI_APPLY_SETTINGS.

Register: WIFI_GATEWAY_DEFAULT

Address: 49254

```
property wifi_ip: str
```

Read the current IP address of WiFi.

Register: WIFI_IP

Address: 49200

```
property wifi_ip_default: str
```

The new IP address of WiFi. Use WIFI_APPLY_SETTINGS.

Register: WIFI_IP_DEFAULT

Address: 49250

```
property wifi_mac: str
```

The MAC address of the WiFi module.

Register: WIFI_MAC

Address: 60024

```
property wifi_password_default: str
```

Write the password for the WiFi network, then use WIFI_APPLY_SETTINGS.

Register: WIFI_PASSWORD_DEFAULT

Address: 49350

```
property wifi_rssi: float
```

WiFi RSSI (signal strength). Typical values are -40 for very good, and -75 for very weak. The T7 microcontroller only gets a new RSSI value from the WiFi module when WiFi communication occurs.

Register: WIFI_RSSI

Address: 49452

```
property wifi_ssid: str
```

Read the current SSID (network name) of WiFi.

Register: WIFI_SSID

Address: 49300

```
property wifi_ssid_default: str
```

The new SSID (network name) of WiFi. Use WIFI_APPLY_SETTINGS.

Register: WIFI_SSID_DEFAULT

Address: 49325

```
property wifi_status: int
```

Status Codes: ASSOCIATED = 2900, ASSOCIATING = 2901, ASSOCIATION_FAILED = 2902, UNPOWERED = 2903, BOOTING = 2904, START_FAILED = 2905, APPLYING_SETTINGS = 2906, DHCP_STARTED = 2907, OTHER = 2909

Register: WIFI_STATUS

Address: 49450

```
property wifi_subnet: str
```

Read the current subnet of WiFi.

Register: WIFI_SUBNET

Address: 49202

```
property wifi_subnet_default: str
```

The new subnet of WiFi. Use WIFI_APPLY_SETTINGS.

Register: WIFI_SUBNET_DEFAULT

Address: 49252

```
property wifi_udp_discovery_only: int
```

Restricts which Modbus operations are allowed over UDP. When set to 1, only the registers needed for discovering units can be read and any other operation will throw an error.

Register: WIFI_UDP_DISCOVERY_ONLY

Address: 49215

```
property wifi_udp_discovery_only_default: int
```

The Enabled/Disabled state of WIFI_UDP_DISCOVERY_ONLY after a power-cycle to the device.

Register: WIFI_UDP_DISCOVERY_ONLY_DEFAULT

Address: 49265

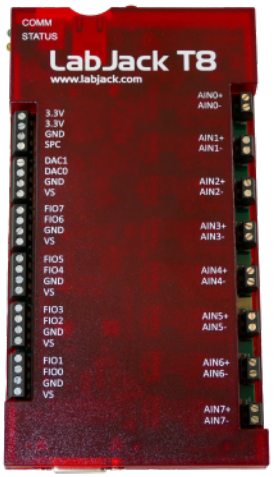
```
property wifi_version: float
```

The current firmware version of the WiFi module, if available.

Register: WIFI_VERSION

Address: 60008

LabJack T8




```
class htf.labjack.T8(*, name: str | None = None, host: str | None = None, port: int = 502,
debuglevel: int = 0, timeout: float = 1.0, retries: int = 0)
```

LabJack adapter for the T8.

Parameters:

- **name** – If name is set the USB device's name is checked against it. If host is None a USB device is opened.
- **host** – If host is set a Modbus-TCP connection is opened. Name must not be set at the same time.
- **port** – The Modbus-TCP port to be used
- **debuglevel** – If set the Modbus packets are printed
- **timeout** – The timeout for every request
- **retries** – The number of retries for Modbus communication

```
property ain0: float
```

Returns the voltage of the specified analog input.

Register: AIN0 (AIN#(0:7)@0)

Address: 0

```
property ain0_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN0_BINARY (AIN#(0:7)_BINARY@0)

Address: 50000

```
property ain0_binary_capture: int
```

T8 Only. Returns the saved 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN0_BINARY_CAPTURE (AIN#(0:7)_BINARY_CAPTURE@0)

Address: 50650

```
property ain0_capture: float
```

T8 Only. Returns the saved voltage of the specified analog input.

Register: AIN0_CAPTURE (AIN#(0:7)_CAPTURE@0)

Address: 650

```
property ain0_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@0)

Address: 9300

```
property ain0_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@0)

Address: 9600

```
property ain0_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@0)

Address: 9900

```
property ain0_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@0)

Address: 10200

```
property ain0_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@0)

Address: 10500

```
property ain0_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@0)

Address: 10800

```
property ain0_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@0)

Address: 11100

```
property ain0_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@0)

Address: 11400

```
property ain0_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@0)

Address: 11700

```
property ain0_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN0_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@0)

Address: 12000

```
property ain0_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN0_EF_INDEX (AIN#(0:149)_EF_INDEX@0)

Address: 9000

```
property ain0_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN0_EF_READ_A (AIN#(0:149)_EF_READ_A@0)

Address: 7000

```
property ain0_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN0_EF_READ_B (AIN#(0:149)_EF_READ_B@0)

Address: 7300

```
property ain0_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN0_EF_READ_C (AIN#(0:149)_EF_READ_C@0)

Address: 7600

```
property ain0_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN0_EF_READ_D (AIN#(0:149)_EF_READ_D@0)

Address: 7900

```
property ain0_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN0_NEGATIVE_CH (AIN#(0:7)_NEGATIVE_CH@0)

Address: 41000

```
property ain0_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN0_RANGE (AIN#(0:7)_RANGE@0)

Address: 40000

```
property ain0_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN0_RESOLUTION_INDEX (AIN#(0:7)_RESOLUTION_INDEX@0)

Address: 41500

```
property ain0_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN0_SETTLING_US (AIN#(0:7)_SETTLING_US@0)

Address: 42000

```
property ain1: float
```

Returns the voltage of the specified analog input.

Register: AIN1 (AIN#(0:7)@1)

Address: 2

```
property ain100_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@100)

Address: 9500

```
property ain100_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@100)

Address: 9800

```
property ain100_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@100)

Address: 10100

```
property ain100_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@100)

Address: 10400

```
property ain100_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@100)

Address: 10700

```
property ain100_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@100)

Address: 11000

```
property ain100_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@100)

Address: 11300

```
property ain100_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@100)

Address: 11600

```
property ain100_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@100)

Address: 11900

```
property ain100_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN100_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@100)

Address: 12200

```
property ain100_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN100_EF_INDEX (AIN#(0:149)_EF_INDEX@100)

Address: 9200

```
property ain100_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN100_EF_READ_A (AIN#(0:149)_EF_READ_A@100)

Address: 7200

```
property ain100_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN100_EF_READ_B (AIN#(0:149)_EF_READ_B@100)

Address: 7500

```
property ain100_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN100_EF_READ_C (AIN#(0:149)_EF_READ_C@100)

Address: 7800

```
property ain100_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN100_EF_READ_D (AIN#(0:149)_EF_READ_D@100)

Address: 8100

```
property ain101_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@101)

Address: 9502

```
property ain101_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@101)

Address: 9802

```
property ain101_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@101)

Address: 10102

```
property ain101_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@101)

Address: 10402

```
property ain101_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@101)

Address: 10702

```
property ain101_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@101)

Address: 11002

```
property ain101_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@101)

Address: 11302

```
property ain101_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@101)

Address: 11602

```
property ain101_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@101)

Address: 11902

```
property ain101_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN101_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@101)

Address: 12202

```
property ain101_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN101_EF_INDEX (AIN#(0:149)_EF_INDEX@101)

Address: 9202

```
property ain101_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN101_EF_READ_A (AIN#(0:149)_EF_READ_A@101)

Address: 7202

```
property ain101_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN101_EF_READ_B (AIN#(0:149)_EF_READ_B@101)

Address: 7502

```
property ain101_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN101_EF_READ_C (AIN#(0:149)_EF_READ_C@101)

Address: 7802

```
property ain101_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN101_EF_READ_D (AIN#(0:149)_EF_READ_D@101)

Address: 8102

```
property ain102_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@102)

Address: 9504

```
property ain102_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@102)

Address: 9804

```
property ain102_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@102)

Address: 10104

```
property ain102_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@102)

Address: 10404

```
property ain102_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@102)

Address: 10704

```
property ain102_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@102)

Address: 11004

```
property ain102_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@102)

Address: 11304

```
property ain102_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@102)

Address: 11604

```
property ain102_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@102)

Address: 11904

```
property ain102_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN102_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@102)

Address: 12204

```
property ain102_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN102_EF_INDEX (AIN#(0:149)_EF_INDEX@102)

Address: 9204

```
property ain102_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN102_EF_READ_A (AIN#(0:149)_EF_READ_A@102)

Address: 7204

```
property ain102_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN102_EF_READ_B (AIN#(0:149)_EF_READ_B@102)

Address: 7504

```
property ain102_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN102_EF_READ_C (AIN#(0:149)_EF_READ_C@102)

Address: 7804

```
property ain102_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN102_EF_READ_D (AIN#(0:149)_EF_READ_D@102)

Address: 8104

```
property ain103_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@103)

Address: 9506

```
property ain103_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@103)

Address: 9806

```
property ain103_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@103)

Address: 10106

```
property ain103_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@103)

Address: 10406

```
property ain103_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@103)

Address: 10706

```
property ain103_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@103)

Address: 11006

```
property ain103_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@103)

Address: 11306

```
property ain103_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@103)

Address: 11606

```
property ain103_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@103)

Address: 11906

```
property ain103_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN103_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@103)

Address: 12206

```
property ain103_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN103_EF_INDEX (AIN#(0:149)_EF_INDEX@103)

Address: 9206

```
property ain103_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN103_EF_READ_A (AIN#(0:149)_EF_READ_A@103)

Address: 7206

```
property ain103_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN103_EF_READ_B (AIN#(0:149)_EF_READ_B@103)

Address: 7506

```
property ain103_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN103_EF_READ_C (AIN#(0:149)_EF_READ_C@103)

Address: 7806

```
property ain103_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN103_EF_READ_D (AIN#(0:149)_EF_READ_D@103)

Address: 8106

```
property ain104_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@104)

Address: 9508

```
property ain104_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@104)

Address: 9808

```
property ain104_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@104)

Address: 10108

```
property ain104_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@104)

Address: 10408

```
property ain104_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@104)

Address: 10708

```
property ain104_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@104)

Address: 11008

```
property ain104_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@104)

Address: 11308

```
property ain104_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@104)

Address: 11608

```
property ain104_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@104)

Address: 11908

```
property ain104_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN104_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@104)

Address: 12208

```
property ain104_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN104_EF_INDEX (AIN#(0:149)_EF_INDEX@104)

Address: 9208

```
property ain104_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN104_EF_READ_A (AIN#(0:149)_EF_READ_A@104)

Address: 7208

```
property ain104_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN104_EF_READ_B (AIN#(0:149)_EF_READ_B@104)

Address: 7508

```
property ain104_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN104_EF_READ_C (AIN#(0:149)_EF_READ_C@104)

Address: 7808

```
property ain104_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN104_EF_READ_D (AIN#(0:149)_EF_READ_D@104)

Address: 8108

```
property ain105_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@105)

Address: 9510

```
property ain105_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@105)

Address: 9810

```
property ain105_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@105)

Address: 10110

```
property ain105_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@105)

Address: 10410

```
property ain105_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@105)

Address: 10710

```
property ain105_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@105)

Address: 11010

```
property ain105_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@105)

Address: 11310

```
property ain105_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@105)

Address: 11610

```
property ain105_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@105)

Address: 11910

```
property ain105_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN105_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@105)

Address: 12210

```
property ain105_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN105_EF_INDEX (AIN#(0:149)_EF_INDEX@105)

Address: 9210

```
property ain105_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN105_EF_READ_A (AIN#(0:149)_EF_READ_A@105)

Address: 7210

```
property ain105_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN105_EF_READ_B (AIN#(0:149)_EF_READ_B@105)

Address: 7510

```
property ain105_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN105_EF_READ_C (AIN#(0:149)_EF_READ_C@105)

Address: 7810

```
property ain105_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN105_EF_READ_D (AIN#(0:149)_EF_READ_D@105)

Address: 8110

```
property ain106_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@106)

Address: 9512

```
property ain106_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@106)

Address: 9812

```
property ain106_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@106)

Address: 10112

```
property ain106_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@106)

Address: 10412

```
property ain106_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@106)

Address: 10712

```
property ain106_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@106)

Address: 11012

```
property ain106_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@106)

Address: 11312

```
property ain106_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@106)

Address: 11612

```
property ain106_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@106)

Address: 11912

```
property ain106_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN106_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@106)

Address: 12212

```
property ain106_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN106_EF_INDEX (AIN#(0:149)_EF_INDEX@106)

Address: 9212

```
property ain106_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN106_EF_READ_A (AIN#(0:149)_EF_READ_A@106)

Address: 7212

```
property ain106_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN106_EF_READ_B (AIN#(0:149)_EF_READ_B@106)

Address: 7512

```
property ain106_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN106_EF_READ_C (AIN#(0:149)_EF_READ_C@106)

Address: 7812

```
property ain106_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN106_EF_READ_D (AIN#(0:149)_EF_READ_D@106)

Address: 8112

```
property ain107_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@107)

Address: 9514

```
property ain107_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@107)

Address: 9814

```
property ain107_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@107)

Address: 10114

```
property ain107_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@107)

Address: 10414

```
property ain107_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@107)

Address: 10714

```
property ain107_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@107)

Address: 11014

```
property ain107_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@107)

Address: 11314

```
property ain107_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@107)

Address: 11614

```
property ain107_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@107)

Address: 11914

```
property ain107_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN107_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@107)

Address: 12214

```
property ain107_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN107_EF_INDEX (AIN#(0:149)_EF_INDEX@107)

Address: 9214

```
property ain107_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN107_EF_READ_A (AIN#(0:149)_EF_READ_A@107)

Address: 7214

```
property ain107_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN107_EF_READ_B (AIN#(0:149)_EF_READ_B@107)

Address: 7514

```
property ain107_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN107_EF_READ_C (AIN#(0:149)_EF_READ_C@107)

Address: 7814

```
property ain107_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN107_EF_READ_D (AIN#(0:149)_EF_READ_D@107)

Address: 8114

```
property ain108_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@108)

Address: 9516

```
property ain108_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@108)

Address: 9816

```
property ain108_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@108)

Address: 10116

```
property ain108_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@108)

Address: 10416

```
property ain108_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@108)

Address: 10716

```
property ain108_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@108)

Address: 11016

```
property ain108_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@108)

Address: 11316

```
property ain108_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@108)

Address: 11616

```
property ain108_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@108)

Address: 11916

```
property ain108_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN108_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@108)

Address: 12216

```
property ain108_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN108_EF_INDEX (AIN#(0:149)_EF_INDEX@108)

Address: 9216

```
property ain108_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN108_EF_READ_A (AIN#(0:149)_EF_READ_A@108)

Address: 7216

```
property ain108_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN108_EF_READ_B (AIN#(0:149)_EF_READ_B@108)

Address: 7516

```
property ain108_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN108_EF_READ_C (AIN#(0:149)_EF_READ_C@108)

Address: 7816

```
property ain108_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN108_EF_READ_D (AIN#(0:149)_EF_READ_D@108)

Address: 8116

```
property ain109_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@109)

Address: 9518

```
property ain109_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@109)

Address: 9818

```
property ain109_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@109)

Address: 10118

```
property ain109_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@109)

Address: 10418

```
property ain109_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@109)

Address: 10718

```
property ain109_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@109)

Address: 11018

```
property ain109_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@109)

Address: 11318

```
property ain109_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@109)

Address: 11618

```
property ain109_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@109)

Address: 11918

```
property ain109_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN109_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@109)

Address: 12218

```
property ain109_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN109_EF_INDEX (AIN#(0:149)_EF_INDEX@109)

Address: 9218

```
property ain109_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN109_EF_READ_A (AIN#(0:149)_EF_READ_A@109)

Address: 7218

```
property ain109_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN109_EF_READ_B (AIN#(0:149)_EF_READ_B@109)

Address: 7518

```
property ain109_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN109_EF_READ_C (AIN#(0:149)_EF_READ_C@109)

Address: 7818

```
property ain109_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN109_EF_READ_D (AIN#(0:149)_EF_READ_D@109)

Address: 8118

```
property ain10_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@10)

Address: 9320

```
property ain10_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@10)

Address: 9620

```
property ain10_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@10)

Address: 9920

```
property ain10_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@10)

Address: 10220

```
property ain10_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@10)

Address: 10520

```
property ain10_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@10)

Address: 10820

```
property ain10_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@10)

Address: 11120

```
property ain10_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@10)

Address: 11420

```
property ain10_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@10)

Address: 11720

```
property ain10_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN10_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@10)

Address: 12020

```
property ain10_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN10_EF_INDEX (AIN#(0:149)_EF_INDEX@10)

Address: 9020

```
property ain10_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN10_EF_READ_A (AIN#(0:149)_EF_READ_A@10)

Address: 7020

```
property ain10_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN10_EF_READ_B (AIN#(0:149)_EF_READ_B@10)

Address: 7320

```
property ain10_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN10_EF_READ_C (AIN#(0:149)_EF_READ_C@10)

Address: 7620

```
property ain10_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN10_EF_READ_D (AIN#(0:149)_EF_READ_D@10)

Address: 7920

```
property ain110_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@110)

Address: 9520

```
property ain110_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@110)

Address: 9820

```
property ain110_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@110)

Address: 10120

```
property ain110_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@110)

Address: 10420

```
property ain110_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@110)

Address: 10720

```
property ain110_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@110)

Address: 11020

```
property ain110_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@110)

Address: 11320

```
property ain110_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@110)

Address: 11620

```
property ain110_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@110)

Address: 11920

```
property ain110_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN110_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@110)

Address: 12220

```
property ain110_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN110_EF_INDEX (AIN#(0:149)_EF_INDEX@110)

Address: 9220

```
property ain110_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN110_EF_READ_A (AIN#(0:149)_EF_READ_A@110)

Address: 7220

```
property ain110_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN110_EF_READ_B (AIN#(0:149)_EF_READ_B@110)

Address: 7520

```
property ain110_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN110_EF_READ_C (AIN#(0:149)_EF_READ_C@110)

Address: 7820

```
property ain110_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN110_EF_READ_D (AIN#(0:149)_EF_READ_D@110)

Address: 8120

```
property ain111_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@111)

Address: 9522

```
property ain111_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@111)

Address: 9822

```
property ain111_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@111)

Address: 10122

```
property ain111_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@111)

Address: 10422

```
property ain111_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@111)

Address: 10722

```
property ain111_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@111)

Address: 11022

```
property ain111_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@111)

Address: 11322

```
property ain111_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@111)

Address: 11622

```
property ain111_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@111)

Address: 11922

```
property ain111_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN111_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@111)

Address: 12222

```
property ain111_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN111_EF_INDEX (AIN#(0:149)_EF_INDEX@111)

Address: 9222

```
property ain111_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN111_EF_READ_A (AIN#(0:149)_EF_READ_A@111)

Address: 7222

```
property ain111_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN111_EF_READ_B (AIN#(0:149)_EF_READ_B@111)

Address: 7522

```
property ain111_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN111_EF_READ_C (AIN#(0:149)_EF_READ_C@111)

Address: 7822

```
property ain111_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN111_EF_READ_D (AIN#(0:149)_EF_READ_D@111)

Address: 8122

```
property ain112_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@112)

Address: 9524

```
property ain112_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@112)

Address: 9824

```
property ain112_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@112)

Address: 10124

```
property ain112_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@112)

Address: 10424

```
property ain112_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@112)

Address: 10724

```
property ain112_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@112)

Address: 11024

```
property ain112_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@112)

Address: 11324

```
property ain112_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@112)

Address: 11624

```
property ain112_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@112)

Address: 11924

```
property ain112_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN112_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@112)

Address: 12224

```
property ain112_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN112_EF_INDEX (AIN#(0:149)_EF_INDEX@112)

Address: 9224

```
property ain112_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN112_EF_READ_A (AIN#(0:149)_EF_READ_A@112)

Address: 7224

```
property ain112_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN112_EF_READ_B (AIN#(0:149)_EF_READ_B@112)

Address: 7524

```
property ain112_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN112_EF_READ_C (AIN#(0:149)_EF_READ_C@112)

Address: 7824

```
property ain112_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN112_EF_READ_D (AIN#(0:149)_EF_READ_D@112)

Address: 8124

```
property ain113_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@113)

Address: 9526

```
property ain113_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@113)

Address: 9826

```
property ain113_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@113)

Address: 10126

```
property ain113_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@113)

Address: 10426

```
property ain113_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@113)

Address: 10726

```
property ain113_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@113)

Address: 11026

```
property ain113_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@113)

Address: 11326

```
property ain113_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@113)

Address: 11626

```
property ain113_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@113)

Address: 11926

```
property ain113_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN113_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@113)

Address: 12226

```
property ain113_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN113_EF_INDEX (AIN#(0:149)_EF_INDEX@113)

Address: 9226

```
property ain113_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN113_EF_READ_A (AIN#(0:149)_EF_READ_A@113)

Address: 7226

```
property ain113_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN113_EF_READ_B (AIN#(0:149)_EF_READ_B@113)

Address: 7526

```
property ain113_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN113_EF_READ_C (AIN#(0:149)_EF_READ_C@113)

Address: 7826

```
property ain113_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN113_EF_READ_D (AIN#(0:149)_EF_READ_D@113)

Address: 8126

```
property ain114_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@114)

Address: 9528

```
property ain114_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@114)

Address: 9828

```
property ain114_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@114)

Address: 10128

```
property ain114_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@114)

Address: 10428

```
property ain114_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@114)

Address: 10728

```
property ain114_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@114)

Address: 11028

```
property ain114_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@114)

Address: 11328

```
property ain114_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@114)

Address: 11628

```
property ain114_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@114)

Address: 11928

```
property ain114_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN114_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@114)

Address: 12228

```
property ain114_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN114_EF_INDEX (AIN#(0:149)_EF_INDEX@114)

Address: 9228

```
property ain114_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN114_EF_READ_A (AIN#(0:149)_EF_READ_A@114)

Address: 7228

```
property ain114_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN114_EF_READ_B (AIN#(0:149)_EF_READ_B@114)

Address: 7528

```
property ain114_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN114_EF_READ_C (AIN#(0:149)_EF_READ_C@114)

Address: 7828

```
property ain114_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN114_EF_READ_D (AIN#(0:149)_EF_READ_D@114)

Address: 8128

```
property ain115_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@115)

Address: 9530

```
property ain115_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@115)

Address: 9830

```
property ain115_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@115)

Address: 10130

```
property ain115_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@115)

Address: 10430

```
property ain115_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@115)

Address: 10730

```
property ain115_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@115)

Address: 11030

```
property ain115_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@115)

Address: 11330

```
property ain115_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@115)

Address: 11630

```
property ain115_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@115)

Address: 11930

```
property ain115_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN115_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@115)

Address: 12230

```
property ain115_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN115_EF_INDEX (AIN#(0:149)_EF_INDEX@115)

Address: 9230

```
property ain115_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN115_EF_READ_A (AIN#(0:149)_EF_READ_A@115)

Address: 7230

```
property ain115_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN115_EF_READ_B (AIN#(0:149)_EF_READ_B@115)

Address: 7530

```
property ain115_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN115_EF_READ_C (AIN#(0:149)_EF_READ_C@115)

Address: 7830

```
property ain115_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN115_EF_READ_D (AIN#(0:149)_EF_READ_D@115)

Address: 8130

```
property ain116_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@116)

Address: 9532

```
property ain116_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@116)

Address: 9832

```
property ain116_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@116)

Address: 10132

```
property ain116_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@116)

Address: 10432

```
property ain116_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@116)

Address: 10732

```
property ain116_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@116)

Address: 11032

```
property ain116_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@116)

Address: 11332

```
property ain116_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@116)

Address: 11632

```
property ain116_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@116)

Address: 11932

```
property ain116_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN116_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@116)

Address: 12232

```
property ain116_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN116_EF_INDEX (AIN#(0:149)_EF_INDEX@116)

Address: 9232

```
property ain116_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN116_EF_READ_A (AIN#(0:149)_EF_READ_A@116)

Address: 7232

```
property ain116_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN116_EF_READ_B (AIN#(0:149)_EF_READ_B@116)

Address: 7532

```
property ain116_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN116_EF_READ_C (AIN#(0:149)_EF_READ_C@116)

Address: 7832

```
property ain116_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN116_EF_READ_D (AIN#(0:149)_EF_READ_D@116)

Address: 8132

```
property ain117_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@117)

Address: 9534

```
property ain117_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@117)

Address: 9834

```
property ain117_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@117)

Address: 10134

```
property ain117_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@117)

Address: 10434

```
property ain117_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@117)

Address: 10734

```
property ain117_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@117)

Address: 11034

```
property ain117_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@117)

Address: 11334

```
property ain117_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@117)

Address: 11634

```
property ain117_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@117)

Address: 11934

```
property ain117_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN117_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@117)

Address: 12234

```
property ain117_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN117_EF_INDEX (AIN#(0:149)_EF_INDEX@117)

Address: 9234

```
property ain117_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN117_EF_READ_A (AIN#(0:149)_EF_READ_A@117)

Address: 7234

```
property ain117_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN117_EF_READ_B (AIN#(0:149)_EF_READ_B@117)

Address: 7534

```
property ain117_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN117_EF_READ_C (AIN#(0:149)_EF_READ_C@117)

Address: 7834

```
property ain117_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN117_EF_READ_D (AIN#(0:149)_EF_READ_D@117)

Address: 8134

```
property ain118_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@118)

Address: 9536

```
property ain118_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@118)

Address: 9836

```
property ain118_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@118)

Address: 10136

```
property ain118_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@118)

Address: 10436

```
property ain118_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@118)

Address: 10736

```
property ain118_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@118)

Address: 11036

```
property ain118_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@118)

Address: 11336

```
property ain118_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@118)

Address: 11636

```
property ain118_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@118)

Address: 11936

```
property ain118_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN118_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@118)

Address: 12236

```
property ain118_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN118_EF_INDEX (AIN#(0:149)_EF_INDEX@118)

Address: 9236

```
property ain118_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN118_EF_READ_A (AIN#(0:149)_EF_READ_A@118)

Address: 7236

```
property ain118_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN118_EF_READ_B (AIN#(0:149)_EF_READ_B@118)

Address: 7536

```
property ain118_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN118_EF_READ_C (AIN#(0:149)_EF_READ_C@118)

Address: 7836

```
property ain118_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN118_EF_READ_D (AIN#(0:149)_EF_READ_D@118)

Address: 8136

```
property ain119_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@119)

Address: 9538

```
property ain119_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@119)

Address: 9838

```
property ain119_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@119)

Address: 10138

```
property ain119_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@119)

Address: 10438

```
property ain119_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@119)

Address: 10738

```
property ain119_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@119)

Address: 11038

```
property ain119_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@119)

Address: 11338

```
property ain119_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@119)

Address: 11638

```
property ain119_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@119)

Address: 11938

```
property ain119_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN119_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@119)

Address: 12238

```
property ain119_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN119_EF_INDEX (AIN#(0:149)_EF_INDEX@119)

Address: 9238

```
property ain119_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN119_EF_READ_A (AIN#(0:149)_EF_READ_A@119)

Address: 7238

```
property ain119_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN119_EF_READ_B (AIN#(0:149)_EF_READ_B@119)

Address: 7538

```
property ain119_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN119_EF_READ_C (AIN#(0:149)_EF_READ_C@119)

Address: 7838

```
property ain119_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN119_EF_READ_D (AIN#(0:149)_EF_READ_D@119)

Address: 8138

```
property ain11_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@11)

Address: 9322

```
property ain11_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@11)

Address: 9622

```
property ain11_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@11)

Address: 9922

```
property ain11_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@11)

Address: 10222

```
property ain11_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@11)

Address: 10522

```
property ain11_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@11)

Address: 10822

```
property ain11_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@11)

Address: 11122

```
property ain11_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@11)

Address: 11422

```
property ain11_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@11)

Address: 11722

```
property ain11_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN11_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@11)

Address: 12022

```
property ain11_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN11_EF_INDEX (AIN#(0:149)_EF_INDEX@11)

Address: 9022

```
property ain11_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN11_EF_READ_A (AIN#(0:149)_EF_READ_A@11)

Address: 7022

```
property ain11_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN11_EF_READ_B (AIN#(0:149)_EF_READ_B@11)

Address: 7322

```
property ain11_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN11_EF_READ_C (AIN#(0:149)_EF_READ_C@11)

Address: 7622

```
property ain11_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN11_EF_READ_D (AIN#(0:149)_EF_READ_D@11)

Address: 7922

```
property ain120_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@120)

Address: 9540

```
property ain120_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@120)

Address: 9840

```
property ain120_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@120)

Address: 10140

```
property ain120_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@120)

Address: 10440

```
property ain120_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@120)

Address: 10740

```
property ain120_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@120)

Address: 11040

```
property ain120_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@120)

Address: 11340

```
property ain120_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@120)

Address: 11640

```
property ain120_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@120)

Address: 11940

```
property ain120_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN120_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@120)

Address: 12240

```
property ain120_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN120_EF_INDEX (AIN#(0:149)_EF_INDEX@120)

Address: 9240

```
property ain120_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN120_EF_READ_A (AIN#(0:149)_EF_READ_A@120)

Address: 7240

```
property ain120_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN120_EF_READ_B (AIN#(0:149)_EF_READ_B@120)

Address: 7540

```
property ain120_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN120_EF_READ_C (AIN#(0:149)_EF_READ_C@120)

Address: 7840

```
property ain120_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN120_EF_READ_D (AIN#(0:149)_EF_READ_D@120)

Address: 8140

```
property ain121_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@121)

Address: 9542

```
property ain121_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@121)

Address: 9842

```
property ain121_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@121)

Address: 10142

```
property ain121_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@121)

Address: 10442

```
property ain121_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@121)

Address: 10742

```
property ain121_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@121)

Address: 11042

```
property ain121_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@121)

Address: 11342

```
property ain121_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@121)

Address: 11642

```
property ain121_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@121)

Address: 11942

```
property ain121_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN121_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@121)

Address: 12242

```
property ain121_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN121_EF_INDEX (AIN#(0:149)_EF_INDEX@121)

Address: 9242

```
property ain121_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN121_EF_READ_A (AIN#(0:149)_EF_READ_A@121)

Address: 7242

```
property ain121_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN121_EF_READ_B (AIN#(0:149)_EF_READ_B@121)

Address: 7542

```
property ain121_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN121_EF_READ_C (AIN#(0:149)_EF_READ_C@121)

Address: 7842

```
property ain121_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN121_EF_READ_D (AIN#(0:149)_EF_READ_D@121)

Address: 8142

```
property ain122_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@122)

Address: 9544

```
property ain122_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@122)

Address: 9844

```
property ain122_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@122)

Address: 10144

```
property ain122_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@122)

Address: 10444

```
property ain122_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@122)

Address: 10744

```
property ain122_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@122)

Address: 11044

```
property ain122_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@122)

Address: 11344

```
property ain122_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@122)

Address: 11644

```
property ain122_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@122)

Address: 11944

```
property ain122_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN122_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@122)

Address: 12244

```
property ain122_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN122_EF_INDEX (AIN#(0:149)_EF_INDEX@122)

Address: 9244

```
property ain122_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN122_EF_READ_A (AIN#(0:149)_EF_READ_A@122)

Address: 7244

```
property ain122_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN122_EF_READ_B (AIN#(0:149)_EF_READ_B@122)

Address: 7544

```
property ain122_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN122_EF_READ_C (AIN#(0:149)_EF_READ_C@122)

Address: 7844

```
property ain122_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN122_EF_READ_D (AIN#(0:149)_EF_READ_D@122)

Address: 8144

```
property ain123_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@123)

Address: 9546

```
property ain123_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@123)

Address: 9846

```
property ain123_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@123)

Address: 10146

```
property ain123_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@123)

Address: 10446

```
property ain123_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@123)

Address: 10746

```
property ain123_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@123)

Address: 11046

```
property ain123_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@123)

Address: 11346

```
property ain123_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@123)

Address: 11646

```
property ain123_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@123)

Address: 11946

```
property ain123_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN123_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@123)

Address: 12246

```
property ain123_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN123_EF_INDEX (AIN#(0:149)_EF_INDEX@123)

Address: 9246

```
property ain123_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN123_EF_READ_A (AIN#(0:149)_EF_READ_A@123)

Address: 7246

```
property ain123_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN123_EF_READ_B (AIN#(0:149)_EF_READ_B@123)

Address: 7546

```
property ain123_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN123_EF_READ_C (AIN#(0:149)_EF_READ_C@123)

Address: 7846

```
property ain123_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN123_EF_READ_D (AIN#(0:149)_EF_READ_D@123)

Address: 8146

```
property ain124_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@124)

Address: 9548

```
property ain124_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@124)

Address: 9848

```
property ain124_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@124)

Address: 10148

```
property ain124_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@124)

Address: 10448

```
property ain124_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@124)

Address: 10748

```
property ain124_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@124)

Address: 11048

```
property ain124_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@124)

Address: 11348

```
property ain124_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@124)

Address: 11648

```
property ain124_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@124)

Address: 11948

```
property ain124_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN124_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@124)

Address: 12248

```
property ain124_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN124_EF_INDEX (AIN#(0:149)_EF_INDEX@124)

Address: 9248

```
property ain124_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN124_EF_READ_A (AIN#(0:149)_EF_READ_A@124)

Address: 7248

```
property ain124_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN124_EF_READ_B (AIN#(0:149)_EF_READ_B@124)

Address: 7548

```
property ain124_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN124_EF_READ_C (AIN#(0:149)_EF_READ_C@124)

Address: 7848

```
property ain124_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN124_EF_READ_D (AIN#(0:149)_EF_READ_D@124)

Address: 8148

```
property ain125_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@125)

Address: 9550

```
property ain125_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@125)

Address: 9850

```
property ain125_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@125)

Address: 10150

```
property ain125_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@125)

Address: 10450

```
property ain125_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@125)

Address: 10750

```
property ain125_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@125)

Address: 11050

```
property ain125_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@125)

Address: 11350

```
property ain125_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@125)

Address: 11650

```
property ain125_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@125)

Address: 11950

```
property ain125_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN125_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@125)

Address: 12250

```
property ain125_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN125_EF_INDEX (AIN#(0:149)_EF_INDEX@125)

Address: 9250

```
property ain125_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN125_EF_READ_A (AIN#(0:149)_EF_READ_A@125)

Address: 7250

```
property ain125_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN125_EF_READ_B (AIN#(0:149)_EF_READ_B@125)

Address: 7550

```
property ain125_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN125_EF_READ_C (AIN#(0:149)_EF_READ_C@125)

Address: 7850

```
property ain125_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN125_EF_READ_D (AIN#(0:149)_EF_READ_D@125)

Address: 8150

```
property ain126_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@126)

Address: 9552

```
property ain126_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@126)

Address: 9852

```
property ain126_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@126)

Address: 10152

```
property ain126_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@126)

Address: 10452

```
property ain126_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@126)

Address: 10752

```
property ain126_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@126)

Address: 11052

```
property ain126_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@126)

Address: 11352

```
property ain126_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@126)

Address: 11652

```
property ain126_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@126)

Address: 11952

```
property ain126_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN126_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@126)

Address: 12252

```
property ain126_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN126_EF_INDEX (AIN#(0:149)_EF_INDEX@126)

Address: 9252

```
property ain126_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN126_EF_READ_A (AIN#(0:149)_EF_READ_A@126)

Address: 7252

```
property ain126_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN126_EF_READ_B (AIN#(0:149)_EF_READ_B@126)

Address: 7552

```
property ain126_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN126_EF_READ_C (AIN#(0:149)_EF_READ_C@126)

Address: 7852

```
property ain126_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN126_EF_READ_D (AIN#(0:149)_EF_READ_D@126)

Address: 8152

```
property ain127_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@127)

Address: 9554

```
property ain127_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@127)

Address: 9854

```
property ain127_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@127)

Address: 10154

```
property ain127_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@127)

Address: 10454

```
property ain127_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@127)

Address: 10754

```
property ain127_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@127)

Address: 11054

```
property ain127_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@127)

Address: 11354

```
property ain127_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@127)

Address: 11654

```
property ain127_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@127)

Address: 11954

```
property ain127_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN127_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@127)

Address: 12254

```
property ain127_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN127_EF_INDEX (AIN#(0:149)_EF_INDEX@127)

Address: 9254

```
property ain127_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN127_EF_READ_A (AIN#(0:149)_EF_READ_A@127)

Address: 7254

```
property ain127_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN127_EF_READ_B (AIN#(0:149)_EF_READ_B@127)

Address: 7554

```
property ain127_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN127_EF_READ_C (AIN#(0:149)_EF_READ_C@127)

Address: 7854

```
property ain127_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN127_EF_READ_D (AIN#(0:149)_EF_READ_D@127)

Address: 8154

```
property ain128_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@128)

Address: 9556

```
property ain128_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@128)

Address: 9856

```
property ain128_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@128)

Address: 10156

```
property ain128_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@128)

Address: 10456

```
property ain128_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@128)

Address: 10756

```
property ain128_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@128)

Address: 11056

```
property ain128_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@128)

Address: 11356

```
property ain128_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@128)

Address: 11656

```
property ain128_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@128)

Address: 11956

```
property ain128_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN128_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@128)

Address: 12256

```
property ain128_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN128_EF_INDEX (AIN#(0:149)_EF_INDEX@128)

Address: 9256

```
property ain128_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN128_EF_READ_A (AIN#(0:149)_EF_READ_A@128)

Address: 7256

```
property ain128_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN128_EF_READ_B (AIN#(0:149)_EF_READ_B@128)

Address: 7556

```
property ain128_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN128_EF_READ_C (AIN#(0:149)_EF_READ_C@128)

Address: 7856

```
property ain128_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN128_EF_READ_D (AIN#(0:149)_EF_READ_D@128)

Address: 8156

```
property ain129_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@129)

Address: 9558

```
property ain129_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@129)

Address: 9858

```
property ain129_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@129)

Address: 10158

```
property ain129_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@129)

Address: 10458

```
property ain129_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@129)

Address: 10758

```
property ain129_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@129)

Address: 11058

```
property ain129_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@129)

Address: 11358

```
property ain129_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@129)

Address: 11658

```
property ain129_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@129)

Address: 11958

```
property ain129_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN129_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@129)

Address: 12258

```
property ain129_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN129_EF_INDEX (AIN#(0:149)_EF_INDEX@129)

Address: 9258

```
property ain129_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN129_EF_READ_A (AIN#(0:149)_EF_READ_A@129)

Address: 7258

```
property ain129_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN129_EF_READ_B (AIN#(0:149)_EF_READ_B@129)

Address: 7558

```
property ain129_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN129_EF_READ_C (AIN#(0:149)_EF_READ_C@129)

Address: 7858

```
property ain129_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN129_EF_READ_D (AIN#(0:149)_EF_READ_D@129)

Address: 8158

```
property ain12_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@12)

Address: 9324

```
property ain12_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@12)

Address: 9624

```
property ain12_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@12)

Address: 9924

```
property ain12_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@12)

Address: 10224

```
property ain12_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@12)

Address: 10524

```
property ain12_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@12)

Address: 10824

```
property ain12_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@12)

Address: 11124

```
property ain12_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@12)

Address: 11424

```
property ain12_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@12)

Address: 11724

```
property ain12_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN12_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@12)

Address: 12024

```
property ain12_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN12_EF_INDEX (AIN#(0:149)_EF_INDEX@12)

Address: 9024

```
property ain12_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN12_EF_READ_A (AIN#(0:149)_EF_READ_A@12)

Address: 7024

```
property ain12_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN12_EF_READ_B (AIN#(0:149)_EF_READ_B@12)

Address: 7324

```
property ain12_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN12_EF_READ_C (AIN#(0:149)_EF_READ_C@12)

Address: 7624

```
property ain12_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN12_EF_READ_D (AIN#(0:149)_EF_READ_D@12)

Address: 7924

```
property ain130_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@130)

Address: 9560

```
property ain130_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@130)

Address: 9860

```
property ain130_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@130)

Address: 10160

```
property ain130_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@130)

Address: 10460

```
property ain130_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@130)

Address: 10760

```
property ain130_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@130)

Address: 11060

```
property ain130_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@130)

Address: 11360

```
property ain130_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@130)

Address: 11660

```
property ain130_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@130)

Address: 11960

```
property ain130_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN130_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@130)

Address: 12260

```
property ain130_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN130_EF_INDEX (AIN#(0:149)_EF_INDEX@130)

Address: 9260

```
property ain130_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN130_EF_READ_A (AIN#(0:149)_EF_READ_A@130)

Address: 7260

```
property ain130_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN130_EF_READ_B (AIN#(0:149)_EF_READ_B@130)

Address: 7560

```
property ain130_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN130_EF_READ_C (AIN#(0:149)_EF_READ_C@130)

Address: 7860

```
property ain130_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN130_EF_READ_D (AIN#(0:149)_EF_READ_D@130)

Address: 8160

```
property ain131_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@131)

Address: 9562

```
property ain131_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@131)

Address: 9862

```
property ain131_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@131)

Address: 10162

```
property ain131_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@131)

Address: 10462

```
property ain131_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@131)

Address: 10762

```
property ain131_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@131)

Address: 11062

```
property ain131_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@131)

Address: 11362

```
property ain131_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@131)

Address: 11662

```
property ain131_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@131)

Address: 11962

```
property ain131_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN131_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@131)

Address: 12262

```
property ain131_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN131_EF_INDEX (AIN#(0:149)_EF_INDEX@131)

Address: 9262

```
property ain131_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN131_EF_READ_A (AIN#(0:149)_EF_READ_A@131)

Address: 7262

```
property ain131_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN131_EF_READ_B (AIN#(0:149)_EF_READ_B@131)

Address: 7562

```
property ain131_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN131_EF_READ_C (AIN#(0:149)_EF_READ_C@131)

Address: 7862

```
property ain131_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN131_EF_READ_D (AIN#(0:149)_EF_READ_D@131)

Address: 8162

```
property ain132_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@132)

Address: 9564

```
property ain132_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@132)

Address: 9864

```
property ain132_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@132)

Address: 10164

```
property ain132_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@132)

Address: 10464

```
property ain132_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@132)

Address: 10764

```
property ain132_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@132)

Address: 11064

```
property ain132_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@132)

Address: 11364

```
property ain132_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@132)

Address: 11664

```
property ain132_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@132)

Address: 11964

```
property ain132_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN132_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@132)

Address: 12264

```
property ain132_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN132_EF_INDEX (AIN#(0:149)_EF_INDEX@132)

Address: 9264

```
property ain132_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN132_EF_READ_A (AIN#(0:149)_EF_READ_A@132)

Address: 7264

```
property ain132_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN132_EF_READ_B (AIN#(0:149)_EF_READ_B@132)

Address: 7564

```
property ain132_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN132_EF_READ_C (AIN#(0:149)_EF_READ_C@132)

Address: 7864

```
property ain132_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN132_EF_READ_D (AIN#(0:149)_EF_READ_D@132)

Address: 8164

```
property ain133_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@133)

Address: 9566

```
property ain133_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@133)

Address: 9866

```
property ain133_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@133)

Address: 10166

```
property ain133_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@133)

Address: 10466

```
property ain133_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@133)

Address: 10766

```
property ain133_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@133)

Address: 11066

```
property ain133_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@133)

Address: 11366

```
property ain133_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@133)

Address: 11666

```
property ain133_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@133)

Address: 11966

```
property ain133_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN133_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@133)

Address: 12266

```
property ain133_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN133_EF_INDEX (AIN#(0:149)_EF_INDEX@133)

Address: 9266

```
property ain133_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN133_EF_READ_A (AIN#(0:149)_EF_READ_A@133)

Address: 7266

```
property ain133_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN133_EF_READ_B (AIN#(0:149)_EF_READ_B@133)

Address: 7566

```
property ain133_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN133_EF_READ_C (AIN#(0:149)_EF_READ_C@133)

Address: 7866

```
property ain133_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN133_EF_READ_D (AIN#(0:149)_EF_READ_D@133)

Address: 8166

```
property ain134_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@134)

Address: 9568

```
property ain134_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@134)

Address: 9868

```
property ain134_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@134)

Address: 10168

```
property ain134_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@134)

Address: 10468

```
property ain134_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@134)

Address: 10768

```
property ain134_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@134)

Address: 11068

```
property ain134_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@134)

Address: 11368

```
property ain134_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@134)

Address: 11668

```
property ain134_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@134)

Address: 11968

```
property ain134_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN134_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@134)

Address: 12268

```
property ain134_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN134_EF_INDEX (AIN#(0:149)_EF_INDEX@134)

Address: 9268

```
property ain134_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN134_EF_READ_A (AIN#(0:149)_EF_READ_A@134)

Address: 7268

```
property ain134_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN134_EF_READ_B (AIN#(0:149)_EF_READ_B@134)

Address: 7568

```
property ain134_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN134_EF_READ_C (AIN#(0:149)_EF_READ_C@134)

Address: 7868

```
property ain134_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN134_EF_READ_D (AIN#(0:149)_EF_READ_D@134)

Address: 8168

```
property ain135_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@135)

Address: 9570

```
property ain135_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@135)

Address: 9870

```
property ain135_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@135)

Address: 10170

```
property ain135_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@135)

Address: 10470

```
property ain135_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@135)

Address: 10770

```
property ain135_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@135)

Address: 11070

```
property ain135_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@135)

Address: 11370

```
property ain135_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@135)

Address: 11670

```
property ain135_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@135)

Address: 11970

```
property ain135_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN135_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@135)

Address: 12270

```
property ain135_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN135_EF_INDEX (AIN#(0:149)_EF_INDEX@135)

Address: 9270

```
property ain135_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN135_EF_READ_A (AIN#(0:149)_EF_READ_A@135)

Address: 7270

```
property ain135_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN135_EF_READ_B (AIN#(0:149)_EF_READ_B@135)

Address: 7570

```
property ain135_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN135_EF_READ_C (AIN#(0:149)_EF_READ_C@135)

Address: 7870

```
property ain135_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN135_EF_READ_D (AIN#(0:149)_EF_READ_D@135)

Address: 8170

```
property ain136_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@136)

Address: 9572

```
property ain136_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@136)

Address: 9872

```
property ain136_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@136)

Address: 10172

```
property ain136_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@136)

Address: 10472

```
property ain136_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@136)

Address: 10772

```
property ain136_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@136)

Address: 11072

```
property ain136_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@136)

Address: 11372

```
property ain136_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@136)

Address: 11672

```
property ain136_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@136)

Address: 11972

```
property ain136_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN136_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@136)

Address: 12272

```
property ain136_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN136_EF_INDEX (AIN#(0:149)_EF_INDEX@136)

Address: 9272

```
property ain136_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN136_EF_READ_A (AIN#(0:149)_EF_READ_A@136)

Address: 7272

```
property ain136_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN136_EF_READ_B (AIN#(0:149)_EF_READ_B@136)

Address: 7572

```
property ain136_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN136_EF_READ_C (AIN#(0:149)_EF_READ_C@136)

Address: 7872

```
property ain136_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN136_EF_READ_D (AIN#(0:149)_EF_READ_D@136)

Address: 8172

```
property ain137_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@137)

Address: 9574

```
property ain137_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@137)

Address: 9874

```
property ain137_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@137)

Address: 10174

```
property ain137_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@137)

Address: 10474

```
property ain137_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@137)

Address: 10774

```
property ain137_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@137)

Address: 11074

```
property ain137_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@137)

Address: 11374

```
property ain137_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@137)

Address: 11674

```
property ain137_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@137)

Address: 11974

```
property ain137_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN137_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@137)

Address: 12274

```
property ain137_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN137_EF_INDEX (AIN#(0:149)_EF_INDEX@137)

Address: 9274

```
property ain137_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN137_EF_READ_A (AIN#(0:149)_EF_READ_A@137)

Address: 7274

```
property ain137_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN137_EF_READ_B (AIN#(0:149)_EF_READ_B@137)

Address: 7574

```
property ain137_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN137_EF_READ_C (AIN#(0:149)_EF_READ_C@137)

Address: 7874

```
property ain137_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN137_EF_READ_D (AIN#(0:149)_EF_READ_D@137)

Address: 8174

```
property ain138_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@138)

Address: 9576

```
property ain138_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@138)

Address: 9876

```
property ain138_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@138)

Address: 10176

```
property ain138_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@138)

Address: 10476

```
property ain138_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@138)

Address: 10776

```
property ain138_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@138)

Address: 11076

```
property ain138_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@138)

Address: 11376

```
property ain138_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@138)

Address: 11676

```
property ain138_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@138)

Address: 11976

```
property ain138_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN138_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@138)

Address: 12276

```
property ain138_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN138_EF_INDEX (AIN#(0:149)_EF_INDEX@138)

Address: 9276

```
property ain138_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN138_EF_READ_A (AIN#(0:149)_EF_READ_A@138)

Address: 7276

```
property ain138_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN138_EF_READ_B (AIN#(0:149)_EF_READ_B@138)

Address: 7576

```
property ain138_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN138_EF_READ_C (AIN#(0:149)_EF_READ_C@138)

Address: 7876

```
property ain138_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN138_EF_READ_D (AIN#(0:149)_EF_READ_D@138)

Address: 8176

```
property ain139_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@139)

Address: 9578

```
property ain139_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@139)

Address: 9878

```
property ain139_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@139)

Address: 10178

```
property ain139_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@139)

Address: 10478

```
property ain139_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@139)

Address: 10778

```
property ain139_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@139)

Address: 11078

```
property ain139_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@139)

Address: 11378

```
property ain139_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@139)

Address: 11678

```
property ain139_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@139)

Address: 11978

```
property ain139_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN139_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@139)

Address: 12278

```
property ain139_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN139_EF_INDEX (AIN#(0:149)_EF_INDEX@139)

Address: 9278

```
property ain139_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN139_EF_READ_A (AIN#(0:149)_EF_READ_A@139)

Address: 7278

```
property ain139_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN139_EF_READ_B (AIN#(0:149)_EF_READ_B@139)

Address: 7578

```
property ain139_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN139_EF_READ_C (AIN#(0:149)_EF_READ_C@139)

Address: 7878

```
property ain139_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN139_EF_READ_D (AIN#(0:149)_EF_READ_D@139)

Address: 8178

```
property ain13_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@13)

Address: 9326

```
property ain13_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@13)

Address: 9626

```
property ain13_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@13)

Address: 9926

```
property ain13_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@13)

Address: 10226

```
property ain13_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@13)

Address: 10526

```
property ain13_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@13)

Address: 10826

```
property ain13_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@13)

Address: 11126

```
property ain13_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@13)

Address: 11426

```
property ain13_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@13)

Address: 11726

```
property ain13_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN13_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@13)

Address: 12026

```
property ain13_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN13_EF_INDEX (AIN#(0:149)_EF_INDEX@13)

Address: 9026

```
property ain13_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN13_EF_READ_A (AIN#(0:149)_EF_READ_A@13)

Address: 7026

```
property ain13_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN13_EF_READ_B (AIN#(0:149)_EF_READ_B@13)

Address: 7326

```
property ain13_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN13_EF_READ_C (AIN#(0:149)_EF_READ_C@13)

Address: 7626

```
property ain13_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN13_EF_READ_D (AIN#(0:149)_EF_READ_D@13)

Address: 7926

```
property ain140_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@140)

Address: 9580

```
property ain140_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@140)

Address: 9880

```
property ain140_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@140)

Address: 10180

```
property ain140_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@140)

Address: 10480

```
property ain140_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@140)

Address: 10780

```
property ain140_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@140)

Address: 11080

```
property ain140_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@140)

Address: 11380

```
property ain140_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@140)

Address: 11680

```
property ain140_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@140)

Address: 11980

```
property ain140_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN140_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@140)

Address: 12280

```
property ain140_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN140_EF_INDEX (AIN#(0:149)_EF_INDEX@140)

Address: 9280

```
property ain140_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN140_EF_READ_A (AIN#(0:149)_EF_READ_A@140)

Address: 7280

```
property ain140_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN140_EF_READ_B (AIN#(0:149)_EF_READ_B@140)

Address: 7580

```
property ain140_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN140_EF_READ_C (AIN#(0:149)_EF_READ_C@140)

Address: 7880

```
property ain140_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN140_EF_READ_D (AIN#(0:149)_EF_READ_D@140)

Address: 8180

```
property ain141_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@141)

Address: 9582

```
property ain141_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@141)

Address: 9882

```
property ain141_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@141)

Address: 10182

```
property ain141_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@141)

Address: 10482

```
property ain141_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@141)

Address: 10782

```
property ain141_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@141)

Address: 11082

```
property ain141_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@141)

Address: 11382

```
property ain141_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@141)

Address: 11682

```
property ain141_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@141)

Address: 11982

```
property ain141_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN141_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@141)

Address: 12282

```
property ain141_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN141_EF_INDEX (AIN#(0:149)_EF_INDEX@141)

Address: 9282

```
property ain141_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN141_EF_READ_A (AIN#(0:149)_EF_READ_A@141)

Address: 7282

```
property ain141_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN141_EF_READ_B (AIN#(0:149)_EF_READ_B@141)

Address: 7582

```
property ain141_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN141_EF_READ_C (AIN#(0:149)_EF_READ_C@141)

Address: 7882

```
property ain141_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN141_EF_READ_D (AIN#(0:149)_EF_READ_D@141)

Address: 8182

```
property ain142_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@142)

Address: 9584

```
property ain142_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@142)

Address: 9884

```
property ain142_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@142)

Address: 10184

```
property ain142_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@142)

Address: 10484

```
property ain142_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@142)

Address: 10784

```
property ain142_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@142)

Address: 11084

```
property ain142_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@142)

Address: 11384

```
property ain142_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@142)

Address: 11684

```
property ain142_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@142)

Address: 11984

```
property ain142_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN142_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@142)

Address: 12284

```
property ain142_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN142_EF_INDEX (AIN#(0:149)_EF_INDEX@142)

Address: 9284

```
property ain142_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN142_EF_READ_A (AIN#(0:149)_EF_READ_A@142)

Address: 7284

```
property ain142_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN142_EF_READ_B (AIN#(0:149)_EF_READ_B@142)

Address: 7584

```
property ain142_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN142_EF_READ_C (AIN#(0:149)_EF_READ_C@142)

Address: 7884

```
property ain142_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN142_EF_READ_D (AIN#(0:149)_EF_READ_D@142)

Address: 8184

```
property ain143_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@143)

Address: 9586

```
property ain143_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@143)

Address: 9886

```
property ain143_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@143)

Address: 10186

```
property ain143_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@143)

Address: 10486

```
property ain143_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@143)

Address: 10786

```
property ain143_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@143)

Address: 11086

```
property ain143_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@143)

Address: 11386

```
property ain143_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@143)

Address: 11686

```
property ain143_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@143)

Address: 11986

```
property ain143_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN143_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@143)

Address: 12286

```
property ain143_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN143_EF_INDEX (AIN#(0:149)_EF_INDEX@143)

Address: 9286

```
property ain143_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN143_EF_READ_A (AIN#(0:149)_EF_READ_A@143)

Address: 7286

```
property ain143_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN143_EF_READ_B (AIN#(0:149)_EF_READ_B@143)

Address: 7586

```
property ain143_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN143_EF_READ_C (AIN#(0:149)_EF_READ_C@143)

Address: 7886

```
property ain143_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN143_EF_READ_D (AIN#(0:149)_EF_READ_D@143)

Address: 8186

```
property ain144_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@144)

Address: 9588

```
property ain144_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@144)

Address: 9888

```
property ain144_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@144)

Address: 10188

```
property ain144_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@144)

Address: 10488

```
property ain144_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@144)

Address: 10788

```
property ain144_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@144)

Address: 11088

```
property ain144_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@144)

Address: 11388

```
property ain144_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@144)

Address: 11688

```
property ain144_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@144)

Address: 11988

```
property ain144_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN144_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@144)

Address: 12288

```
property ain144_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN144_EF_INDEX (AIN#(0:149)_EF_INDEX@144)

Address: 9288

```
property ain144_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN144_EF_READ_A (AIN#(0:149)_EF_READ_A@144)

Address: 7288

```
property ain144_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN144_EF_READ_B (AIN#(0:149)_EF_READ_B@144)

Address: 7588

```
property ain144_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN144_EF_READ_C (AIN#(0:149)_EF_READ_C@144)

Address: 7888

```
property ain144_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN144_EF_READ_D (AIN#(0:149)_EF_READ_D@144)

Address: 8188

```
property ain145_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@145)

Address: 9590

```
property ain145_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@145)

Address: 9890

```
property ain145_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@145)

Address: 10190

```
property ain145_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@145)

Address: 10490

```
property ain145_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@145)

Address: 10790

```
property ain145_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@145)

Address: 11090

```
property ain145_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@145)

Address: 11390

```
property ain145_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@145)

Address: 11690

```
property ain145_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@145)

Address: 11990

```
property ain145_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN145_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@145)

Address: 12290

```
property ain145_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN145_EF_INDEX (AIN#(0:149)_EF_INDEX@145)

Address: 9290

```
property ain145_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN145_EF_READ_A (AIN#(0:149)_EF_READ_A@145)

Address: 7290

```
property ain145_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN145_EF_READ_B (AIN#(0:149)_EF_READ_B@145)

Address: 7590

```
property ain145_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN145_EF_READ_C (AIN#(0:149)_EF_READ_C@145)

Address: 7890

```
property ain145_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN145_EF_READ_D (AIN#(0:149)_EF_READ_D@145)

Address: 8190

```
property ain146_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@146)

Address: 9592

```
property ain146_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@146)

Address: 9892

```
property ain146_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@146)

Address: 10192

```
property ain146_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@146)

Address: 10492

```
property ain146_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@146)

Address: 10792

```
property ain146_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@146)

Address: 11092

```
property ain146_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@146)

Address: 11392

```
property ain146_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@146)

Address: 11692

```
property ain146_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@146)

Address: 11992

```
property ain146_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN146_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@146)

Address: 12292

```
property ain146_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN146_EF_INDEX (AIN#(0:149)_EF_INDEX@146)

Address: 9292

```
property ain146_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN146_EF_READ_A (AIN#(0:149)_EF_READ_A@146)

Address: 7292

```
property ain146_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN146_EF_READ_B (AIN#(0:149)_EF_READ_B@146)

Address: 7592

```
property ain146_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN146_EF_READ_C (AIN#(0:149)_EF_READ_C@146)

Address: 7892

```
property ain146_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN146_EF_READ_D (AIN#(0:149)_EF_READ_D@146)

Address: 8192

```
property ain147_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@147)

Address: 9594

```
property ain147_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@147)

Address: 9894

```
property ain147_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@147)

Address: 10194

```
property ain147_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@147)

Address: 10494

```
property ain147_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@147)

Address: 10794

```
property ain147_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@147)

Address: 11094

```
property ain147_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@147)

Address: 11394

```
property ain147_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@147)

Address: 11694

```
property ain147_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@147)

Address: 11994

```
property ain147_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN147_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@147)

Address: 12294

```
property ain147_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN147_EF_INDEX (AIN#(0:149)_EF_INDEX@147)

Address: 9294

```
property ain147_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN147_EF_READ_A (AIN#(0:149)_EF_READ_A@147)

Address: 7294

```
property ain147_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN147_EF_READ_B (AIN#(0:149)_EF_READ_B@147)

Address: 7594

```
property ain147_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN147_EF_READ_C (AIN#(0:149)_EF_READ_C@147)

Address: 7894

```
property ain147_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN147_EF_READ_D (AIN#(0:149)_EF_READ_D@147)

Address: 8194

```
property ain148_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@148)

Address: 9596

```
property ain148_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@148)

Address: 9896

```
property ain148_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@148)

Address: 10196

```
property ain148_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@148)

Address: 10496

```
property ain148_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@148)

Address: 10796

```
property ain148_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@148)

Address: 11096

```
property ain148_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@148)

Address: 11396

```
property ain148_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@148)

Address: 11696

```
property ain148_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@148)

Address: 11996

```
property ain148_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN148_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@148)

Address: 12296

```
property ain148_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN148_EF_INDEX (AIN#(0:149)_EF_INDEX@148)

Address: 9296

```
property ain148_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN148_EF_READ_A (AIN#(0:149)_EF_READ_A@148)

Address: 7296

```
property ain148_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN148_EF_READ_B (AIN#(0:149)_EF_READ_B@148)

Address: 7596

```
property ain148_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN148_EF_READ_C (AIN#(0:149)_EF_READ_C@148)

Address: 7896

```
property ain148_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN148_EF_READ_D (AIN#(0:149)_EF_READ_D@148)

Address: 8196

```
property ain149_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@149)

Address: 9598

```
property ain149_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@149)

Address: 9898

```
property ain149_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@149)

Address: 10198

```
property ain149_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@149)

Address: 10498

```
property ain149_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@149)

Address: 10798

```
property ain149_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@149)

Address: 11098

```
property ain149_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@149)

Address: 11398

```
property ain149_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@149)

Address: 11698

```
property ain149_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@149)

Address: 11998

```
property ain149_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN149_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@149)

Address: 12298

```
property ain149_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN149_EF_INDEX (AIN#(0:149)_EF_INDEX@149)

Address: 9298

```
property ain149_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN149_EF_READ_A (AIN#(0:149)_EF_READ_A@149)

Address: 7298

```
property ain149_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN149_EF_READ_B (AIN#(0:149)_EF_READ_B@149)

Address: 7598

```
property ain149_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN149_EF_READ_C (AIN#(0:149)_EF_READ_C@149)

Address: 7898

```
property ain149_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN149_EF_READ_D (AIN#(0:149)_EF_READ_D@149)

Address: 8198

```
property ain14_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@14)

Address: 9328

```
property ain14_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@14)

Address: 9628

```
property ain14_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@14)

Address: 9928

```
property ain14_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@14)

Address: 10228

```
property ain14_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@14)

Address: 10528

```
property ain14_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@14)

Address: 10828

```
property ain14_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@14)

Address: 11128

```
property ain14_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@14)

Address: 11428

```
property ain14_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@14)

Address: 11728

```
property ain14_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN14_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@14)

Address: 12028

```
property ain14_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN14_EF_INDEX (AIN#(0:149)_EF_INDEX@14)

Address: 9028

```
property ain14_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN14_EF_READ_A (AIN#(0:149)_EF_READ_A@14)

Address: 7028

```
property ain14_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN14_EF_READ_B (AIN#(0:149)_EF_READ_B@14)

Address: 7328

```
property ain14_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN14_EF_READ_C (AIN#(0:149)_EF_READ_C@14)

Address: 7628

```
property ain14_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN14_EF_READ_D (AIN#(0:149)_EF_READ_D@14)

Address: 7928

```
property ain15_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@15)

Address: 9330

```
property ain15_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@15)

Address: 9630

```
property ain15_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@15)

Address: 9930

```
property ain15_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@15)

Address: 10230

```
property ain15_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@15)

Address: 10530

```
property ain15_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@15)

Address: 10830

```
property ain15_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@15)

Address: 11130

```
property ain15_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@15)

Address: 11430

```
property ain15_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@15)

Address: 11730

```
property ain15_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN15_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@15)

Address: 12030

```
property ain15_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN15_EF_INDEX (AIN#(0:149)_EF_INDEX@15)

Address: 9030

```
property ain15_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN15_EF_READ_A (AIN#(0:149)_EF_READ_A@15)

Address: 7030

```
property ain15_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN15_EF_READ_B (AIN#(0:149)_EF_READ_B@15)

Address: 7330

```
property ain15_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN15_EF_READ_C (AIN#(0:149)_EF_READ_C@15)

Address: 7630

```
property ain15_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN15_EF_READ_D (AIN#(0:149)_EF_READ_D@15)

Address: 7930

```
property ain16_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@16)

Address: 9332

```
property ain16_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@16)

Address: 9632

```
property ain16_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@16)

Address: 9932

```
property ain16_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@16)

Address: 10232

```
property ain16_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@16)

Address: 10532

```
property ain16_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@16)

Address: 10832

```
property ain16_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@16)

Address: 11132

```
property ain16_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@16)

Address: 11432

```
property ain16_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@16)

Address: 11732

```
property ain16_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN16_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@16)

Address: 12032

```
property ain16_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN16_EF_INDEX (AIN#(0:149)_EF_INDEX@16)

Address: 9032

```
property ain16_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN16_EF_READ_A (AIN#(0:149)_EF_READ_A@16)

Address: 7032

```
property ain16_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN16_EF_READ_B (AIN#(0:149)_EF_READ_B@16)

Address: 7332

```
property ain16_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN16_EF_READ_C (AIN#(0:149)_EF_READ_C@16)

Address: 7632

```
property ain16_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN16_EF_READ_D (AIN#(0:149)_EF_READ_D@16)

Address: 7932

```
property ain17_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@17)

Address: 9334

```
property ain17_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@17)

Address: 9634

```
property ain17_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@17)

Address: 9934

```
property ain17_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@17)

Address: 10234

```
property ain17_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@17)

Address: 10534

```
property ain17_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@17)

Address: 10834

```
property ain17_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@17)

Address: 11134

```
property ain17_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@17)

Address: 11434

```
property ain17_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@17)

Address: 11734

```
property ain17_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN17_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@17)

Address: 12034

```
property ain17_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN17_EF_INDEX (AIN#(0:149)_EF_INDEX@17)

Address: 9034

```
property ain17_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN17_EF_READ_A (AIN#(0:149)_EF_READ_A@17)

Address: 7034

```
property ain17_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN17_EF_READ_B (AIN#(0:149)_EF_READ_B@17)

Address: 7334

```
property ain17_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN17_EF_READ_C (AIN#(0:149)_EF_READ_C@17)

Address: 7634

```
property ain17_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN17_EF_READ_D (AIN#(0:149)_EF_READ_D@17)

Address: 7934

```
property ain18_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@18)

Address: 9336

```
property ain18_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@18)

Address: 9636

```
property ain18_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@18)

Address: 9936

```
property ain18_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@18)

Address: 10236

```
property ain18_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@18)

Address: 10536

```
property ain18_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@18)

Address: 10836

```
property ain18_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@18)

Address: 11136

```
property ain18_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@18)

Address: 11436

```
property ain18_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@18)

Address: 11736

```
property ain18_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN18_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@18)

Address: 12036

```
property ain18_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN18_EF_INDEX (AIN#(0:149)_EF_INDEX@18)

Address: 9036

```
property ain18_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN18_EF_READ_A (AIN#(0:149)_EF_READ_A@18)

Address: 7036

```
property ain18_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN18_EF_READ_B (AIN#(0:149)_EF_READ_B@18)

Address: 7336

```
property ain18_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN18_EF_READ_C (AIN#(0:149)_EF_READ_C@18)

Address: 7636

```
property ain18_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN18_EF_READ_D (AIN#(0:149)_EF_READ_D@18)

Address: 7936

```
property ain19_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@19)

Address: 9338

```
property ain19_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@19)

Address: 9638

```
property ain19_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@19)

Address: 9938

```
property ain19_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@19)

Address: 10238

```
property ain19_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@19)

Address: 10538

```
property ain19_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@19)

Address: 10838

```
property ain19_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@19)

Address: 11138

```
property ain19_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@19)

Address: 11438

```
property ain19_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@19)

Address: 11738

```
property ain19_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN19_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@19)

Address: 12038

```
property ain19_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN19_EF_INDEX (AIN#(0:149)_EF_INDEX@19)

Address: 9038

```
property ain19_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN19_EF_READ_A (AIN#(0:149)_EF_READ_A@19)

Address: 7038

```
property ain19_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN19_EF_READ_B (AIN#(0:149)_EF_READ_B@19)

Address: 7338

```
property ain19_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN19_EF_READ_C (AIN#(0:149)_EF_READ_C@19)

Address: 7638

```
property ain19_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN19_EF_READ_D (AIN#(0:149)_EF_READ_D@19)

Address: 7938

```
property ain1_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN1_BINARY (AIN#(0:7)_BINARY@1)

Address: 50002

```
property ain1_binary_capture: int
```

T8 Only. Returns the saved 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN1_BINARY_CAPTURE (AIN#(0:7)_BINARY_CAPTURE@1)

Address: 50652

```
property ain1_capture: float
```

T8 Only. Returns the saved voltage of the specified analog input.

Register: AIN1_CAPTURE (AIN#(0:7)_CAPTURE@1)

Address: 652

```
property ain1_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@1)

Address: 9302

```
property ain1_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@1)

Address: 9602

```
property ain1_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@1)

Address: 9902

```
property ain1_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@1)

Address: 10202

```
property ain1_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@1)

Address: 10502

```
property ain1_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@1)

Address: 10802

```
property ain1_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@1)

Address: 11102

```
property ain1_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@1)

Address: 11402

```
property ain1_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@1)

Address: 11702

```
property ain1_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN1_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@1)

Address: 12002

```
property ain1_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN1_EF_INDEX (AIN#(0:149)_EF_INDEX@1)

Address: 9002

```
property ain1_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN1_EF_READ_A (AIN#(0:149)_EF_READ_A@1)

Address: 7002

```
property ain1_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN1_EF_READ_B (AIN#(0:149)_EF_READ_B@1)

Address: 7302

```
property ain1_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN1_EF_READ_C (AIN#(0:149)_EF_READ_C@1)

Address: 7602

```
property ain1_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN1_EF_READ_D (AIN#(0:149)_EF_READ_D@1)

Address: 7902

```
property ain1_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN1_NEGATIVE_CH (AIN#(0:7)_NEGATIVE_CH@1)

Address: 41001

```
property ain1_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN1_RANGE (AIN#(0:7)_RANGE@1)

Address: 40002

```
property ain1_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN1_RESOLUTION_INDEX (AIN#(0:7)_RESOLUTION_INDEX@1)

Address: 41501

```
property ain1_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN1_SETTLING_US (AIN#(0:7)_SETTLING_US@1)

Address: 42002

```
property ain2: float
```

Returns the voltage of the specified analog input.

Register: AIN2 (AIN#(0:7)@2)

Address: 4

```
property ain20_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@20)

Address: 9340

```
property ain20_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@20)

Address: 9640

```
property ain20_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@20)

Address: 9940

```
property ain20_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@20)

Address: 10240

```
property ain20_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@20)

Address: 10540

```
property ain20_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@20)

Address: 10840

```
property ain20_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@20)

Address: 11140

```
property ain20_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@20)

Address: 11440

```
property ain20_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@20)

Address: 11740

```
property ain20_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN20_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@20)

Address: 12040

```
property ain20_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN20_EF_INDEX (AIN#(0:149)_EF_INDEX@20)

Address: 9040

```
property ain20_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN20_EF_READ_A (AIN#(0:149)_EF_READ_A@20)

Address: 7040

```
property ain20_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN20_EF_READ_B (AIN#(0:149)_EF_READ_B@20)

Address: 7340

```
property ain20_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN20_EF_READ_C (AIN#(0:149)_EF_READ_C@20)

Address: 7640

```
property ain20_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN20_EF_READ_D (AIN#(0:149)_EF_READ_D@20)

Address: 7940

```
property ain21_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@21)

Address: 9342

```
property ain21_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@21)

Address: 9642

```
property ain21_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@21)

Address: 9942

```
property ain21_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@21)

Address: 10242

```
property ain21_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@21)

Address: 10542

```
property ain21_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@21)

Address: 10842

```
property ain21_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@21)

Address: 11142

```
property ain21_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@21)

Address: 11442

```
property ain21_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@21)

Address: 11742

```
property ain21_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN21_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@21)

Address: 12042

```
property ain21_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN21_EF_INDEX (AIN#(0:149)_EF_INDEX@21)

Address: 9042

```
property ain21_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN21_EF_READ_A (AIN#(0:149)_EF_READ_A@21)

Address: 7042

```
property ain21_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN21_EF_READ_B (AIN#(0:149)_EF_READ_B@21)

Address: 7342

```
property ain21_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN21_EF_READ_C (AIN#(0:149)_EF_READ_C@21)

Address: 7642

```
property ain21_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN21_EF_READ_D (AIN#(0:149)_EF_READ_D@21)

Address: 7942

```
property ain22_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@22)

Address: 9344

```
property ain22_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@22)

Address: 9644

```
property ain22_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@22)

Address: 9944

```
property ain22_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@22)

Address: 10244

```
property ain22_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@22)

Address: 10544

```
property ain22_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@22)

Address: 10844

```
property ain22_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@22)

Address: 11144

```
property ain22_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@22)

Address: 11444

```
property ain22_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@22)

Address: 11744

```
property ain22_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN22_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@22)

Address: 12044

```
property ain22_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN22_EF_INDEX (AIN#(0:149)_EF_INDEX@22)

Address: 9044

```
property ain22_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN22_EF_READ_A (AIN#(0:149)_EF_READ_A@22)

Address: 7044

```
property ain22_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN22_EF_READ_B (AIN#(0:149)_EF_READ_B@22)

Address: 7344

```
property ain22_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN22_EF_READ_C (AIN#(0:149)_EF_READ_C@22)

Address: 7644

```
property ain22_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN22_EF_READ_D (AIN#(0:149)_EF_READ_D@22)

Address: 7944

```
property ain23_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@23)

Address: 9346

```
property ain23_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@23)

Address: 9646

```
property ain23_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@23)

Address: 9946

```
property ain23_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@23)

Address: 10246

```
property ain23_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@23)

Address: 10546

```
property ain23_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@23)

Address: 10846

```
property ain23_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@23)

Address: 11146

```
property ain23_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@23)

Address: 11446

```
property ain23_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@23)

Address: 11746

```
property ain23_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN23_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@23)

Address: 12046

```
property ain23_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN23_EF_INDEX (AIN#(0:149)_EF_INDEX@23)

Address: 9046

```
property ain23_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN23_EF_READ_A (AIN#(0:149)_EF_READ_A@23)

Address: 7046

```
property ain23_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN23_EF_READ_B (AIN#(0:149)_EF_READ_B@23)

Address: 7346

```
property ain23_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN23_EF_READ_C (AIN#(0:149)_EF_READ_C@23)

Address: 7646

```
property ain23_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN23_EF_READ_D (AIN#(0:149)_EF_READ_D@23)

Address: 7946

```
property ain24_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@24)

Address: 9348

```
property ain24_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@24)

Address: 9648

```
property ain24_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@24)

Address: 9948

```
property ain24_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@24)

Address: 10248

```
property ain24_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@24)

Address: 10548

```
property ain24_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@24)

Address: 10848

```
property ain24_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@24)

Address: 11148

```
property ain24_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@24)

Address: 11448

```
property ain24_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@24)

Address: 11748

```
property ain24_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN24_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@24)

Address: 12048

```
property ain24_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN24_EF_INDEX (AIN#(0:149)_EF_INDEX@24)

Address: 9048

```
property ain24_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN24_EF_READ_A (AIN#(0:149)_EF_READ_A@24)

Address: 7048

```
property ain24_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN24_EF_READ_B (AIN#(0:149)_EF_READ_B@24)

Address: 7348

```
property ain24_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN24_EF_READ_C (AIN#(0:149)_EF_READ_C@24)

Address: 7648

```
property ain24_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN24_EF_READ_D (AIN#(0:149)_EF_READ_D@24)

Address: 7948

```
property ain25_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@25)

Address: 9350

```
property ain25_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@25)

Address: 9650

```
property ain25_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@25)

Address: 9950

```
property ain25_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@25)

Address: 10250

```
property ain25_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@25)

Address: 10550

```
property ain25_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@25)

Address: 10850

```
property ain25_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@25)

Address: 11150

```
property ain25_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@25)

Address: 11450

```
property ain25_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@25)

Address: 11750

```
property ain25_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN25_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@25)

Address: 12050

```
property ain25_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN25_EF_INDEX (AIN#(0:149)_EF_INDEX@25)

Address: 9050

```
property ain25_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN25_EF_READ_A (AIN#(0:149)_EF_READ_A@25)

Address: 7050

```
property ain25_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN25_EF_READ_B (AIN#(0:149)_EF_READ_B@25)

Address: 7350

```
property ain25_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN25_EF_READ_C (AIN#(0:149)_EF_READ_C@25)

Address: 7650

```
property ain25_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN25_EF_READ_D (AIN#(0:149)_EF_READ_D@25)

Address: 7950

```
property ain26_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@26)

Address: 9352

```
property ain26_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@26)

Address: 9652

```
property ain26_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@26)

Address: 9952

```
property ain26_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@26)

Address: 10252

```
property ain26_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@26)

Address: 10552

```
property ain26_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@26)

Address: 10852

```
property ain26_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@26)

Address: 11152

```
property ain26_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@26)

Address: 11452

```
property ain26_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@26)

Address: 11752

```
property ain26_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN26_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@26)

Address: 12052

```
property ain26_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN26_EF_INDEX (AIN#(0:149)_EF_INDEX@26)

Address: 9052

```
property ain26_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN26_EF_READ_A (AIN#(0:149)_EF_READ_A@26)

Address: 7052

```
property ain26_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN26_EF_READ_B (AIN#(0:149)_EF_READ_B@26)

Address: 7352

```
property ain26_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN26_EF_READ_C (AIN#(0:149)_EF_READ_C@26)

Address: 7652

```
property ain26_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN26_EF_READ_D (AIN#(0:149)_EF_READ_D@26)

Address: 7952

```
property ain27_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@27)

Address: 9354

```
property ain27_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@27)

Address: 9654

```
property ain27_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@27)

Address: 9954

```
property ain27_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@27)

Address: 10254

```
property ain27_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@27)

Address: 10554

```
property ain27_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@27)

Address: 10854

```
property ain27_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@27)

Address: 11154

```
property ain27_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@27)

Address: 11454

```
property ain27_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@27)

Address: 11754

```
property ain27_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN27_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@27)

Address: 12054

```
property ain27_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN27_EF_INDEX (AIN#(0:149)_EF_INDEX@27)

Address: 9054

```
property ain27_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN27_EF_READ_A (AIN#(0:149)_EF_READ_A@27)

Address: 7054

```
property ain27_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN27_EF_READ_B (AIN#(0:149)_EF_READ_B@27)

Address: 7354

```
property ain27_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN27_EF_READ_C (AIN#(0:149)_EF_READ_C@27)

Address: 7654

```
property ain27_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN27_EF_READ_D (AIN#(0:149)_EF_READ_D@27)

Address: 7954

```
property ain28_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@28)

Address: 9356

```
property ain28_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@28)

Address: 9656

```
property ain28_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@28)

Address: 9956

```
property ain28_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@28)

Address: 10256

```
property ain28_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@28)

Address: 10556

```
property ain28_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@28)

Address: 10856

```
property ain28_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@28)

Address: 11156

```
property ain28_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@28)

Address: 11456

```
property ain28_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@28)

Address: 11756

```
property ain28_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN28_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@28)

Address: 12056

```
property ain28_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN28_EF_INDEX (AIN#(0:149)_EF_INDEX@28)

Address: 9056

```
property ain28_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN28_EF_READ_A (AIN#(0:149)_EF_READ_A@28)

Address: 7056

```
property ain28_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN28_EF_READ_B (AIN#(0:149)_EF_READ_B@28)

Address: 7356

```
property ain28_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN28_EF_READ_C (AIN#(0:149)_EF_READ_C@28)

Address: 7656

```
property ain28_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN28_EF_READ_D (AIN#(0:149)_EF_READ_D@28)

Address: 7956

```
property ain29_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@29)

Address: 9358

```
property ain29_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@29)

Address: 9658

```
property ain29_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@29)

Address: 9958

```
property ain29_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@29)

Address: 10258

```
property ain29_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@29)

Address: 10558

```
property ain29_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@29)

Address: 10858

```
property ain29_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@29)

Address: 11158

```
property ain29_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@29)

Address: 11458

```
property ain29_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@29)

Address: 11758

```
property ain29_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN29_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@29)

Address: 12058

```
property ain29_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN29_EF_INDEX (AIN#(0:149)_EF_INDEX@29)

Address: 9058

```
property ain29_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN29_EF_READ_A (AIN#(0:149)_EF_READ_A@29)

Address: 7058

```
property ain29_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN29_EF_READ_B (AIN#(0:149)_EF_READ_B@29)

Address: 7358

```
property ain29_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN29_EF_READ_C (AIN#(0:149)_EF_READ_C@29)

Address: 7658

```
property ain29_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN29_EF_READ_D (AIN#(0:149)_EF_READ_D@29)

Address: 7958

```
property ain2_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN2_BINARY (AIN#(0:7)_BINARY@2)

Address: 50004

```
property ain2_binary_capture: int
```

T8 Only. Returns the saved 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN2_BINARY_CAPTURE (AIN#(0:7)_BINARY_CAPTURE@2)

Address: 50654

```
property ain2_capture: float
```

T8 Only. Returns the saved voltage of the specified analog input.

Register: AIN2_CAPTURE (AIN#(0:7)_CAPTURE@2)

Address: 654

```
property ain2_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@2)

Address: 9304

```
property ain2_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@2)

Address: 9604

```
property ain2_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@2)

Address: 9904

```
property ain2_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@2)

Address: 10204

```
property ain2_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@2)

Address: 10504

```
property ain2_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@2)

Address: 10804

```
property ain2_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@2)

Address: 11104

```
property ain2_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@2)

Address: 11404

```
property ain2_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@2)

Address: 11704

```
property ain2_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN2_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@2)

Address: 12004

```
property ain2_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN2_EF_INDEX (AIN#(0:149)_EF_INDEX@2)

Address: 9004

```
property ain2_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN2_EF_READ_A (AIN#(0:149)_EF_READ_A@2)

Address: 7004

```
property ain2_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN2_EF_READ_B (AIN#(0:149)_EF_READ_B@2)

Address: 7304

```
property ain2_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN2_EF_READ_C (AIN#(0:149)_EF_READ_C@2)

Address: 7604

```
property ain2_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN2_EF_READ_D (AIN#(0:149)_EF_READ_D@2)

Address: 7904

```
property ain2_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN2_NEGATIVE_CH (AIN#(0:7)_NEGATIVE_CH@2)

Address: 41002

```
property ain2_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN2_RANGE (AIN#(0:7)_RANGE@2)

Address: 40004

```
property ain2_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN2_RESOLUTION_INDEX (AIN#(0:7)_RESOLUTION_INDEX@2)

Address: 41502

```
property ain2_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN2_SETTLING_US (AIN#(0:7)_SETTLING_US@2)

Address: 42004

```
property ain3: float
```

Returns the voltage of the specified analog input.

Register: AIN3 (AIN#(0:7)@3)

Address: 6

```
property ain30_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@30)

Address: 9360

```
property ain30_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@30)

Address: 9660

```
property ain30_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@30)

Address: 9960

```
property ain30_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@30)

Address: 10260

```
property ain30_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@30)

Address: 10560

```
property ain30_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@30)

Address: 10860

```
property ain30_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@30)

Address: 11160

```
property ain30_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@30)

Address: 11460

```
property ain30_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@30)

Address: 11760

```
property ain30_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN30_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@30)

Address: 12060

```
property ain30_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN30_EF_INDEX (AIN#(0:149)_EF_INDEX@30)

Address: 9060

```
property ain30_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN30_EF_READ_A (AIN#(0:149)_EF_READ_A@30)

Address: 7060

```
property ain30_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN30_EF_READ_B (AIN#(0:149)_EF_READ_B@30)

Address: 7360

```
property ain30_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN30_EF_READ_C (AIN#(0:149)_EF_READ_C@30)

Address: 7660

```
property ain30_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN30_EF_READ_D (AIN#(0:149)_EF_READ_D@30)

Address: 7960

```
property ain31_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@31)

Address: 9362

```
property ain31_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@31)

Address: 9662

```
property ain31_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@31)

Address: 9962

```
property ain31_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@31)

Address: 10262

```
property ain31_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@31)

Address: 10562

```
property ain31_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@31)

Address: 10862

```
property ain31_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@31)

Address: 11162

```
property ain31_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@31)

Address: 11462

```
property ain31_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@31)

Address: 11762

```
property ain31_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN31_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@31)

Address: 12062

```
property ain31_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN31_EF_INDEX (AIN#(0:149)_EF_INDEX@31)

Address: 9062

```
property ain31_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN31_EF_READ_A (AIN#(0:149)_EF_READ_A@31)

Address: 7062

```
property ain31_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN31_EF_READ_B (AIN#(0:149)_EF_READ_B@31)

Address: 7362

```
property ain31_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN31_EF_READ_C (AIN#(0:149)_EF_READ_C@31)

Address: 7662

```
property ain31_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN31_EF_READ_D (AIN#(0:149)_EF_READ_D@31)

Address: 7962

```
property ain32_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@32)

Address: 9364

```
property ain32_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@32)

Address: 9664

```
property ain32_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@32)

Address: 9964

```
property ain32_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@32)

Address: 10264

```
property ain32_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@32)

Address: 10564

```
property ain32_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@32)

Address: 10864

```
property ain32_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@32)

Address: 11164

```
property ain32_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@32)

Address: 11464

```
property ain32_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@32)

Address: 11764

```
property ain32_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN32_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@32)

Address: 12064

```
property ain32_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN32_EF_INDEX (AIN#(0:149)_EF_INDEX@32)

Address: 9064

```
property ain32_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN32_EF_READ_A (AIN#(0:149)_EF_READ_A@32)

Address: 7064

```
property ain32_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN32_EF_READ_B (AIN#(0:149)_EF_READ_B@32)

Address: 7364

```
property ain32_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN32_EF_READ_C (AIN#(0:149)_EF_READ_C@32)

Address: 7664

```
property ain32_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN32_EF_READ_D (AIN#(0:149)_EF_READ_D@32)

Address: 7964

```
property ain33_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@33)

Address: 9366

```
property ain33_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@33)

Address: 9666

```
property ain33_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@33)

Address: 9966

```
property ain33_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@33)

Address: 10266

```
property ain33_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@33)

Address: 10566

```
property ain33_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@33)

Address: 10866

```
property ain33_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@33)

Address: 11166

```
property ain33_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@33)

Address: 11466

```
property ain33_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@33)

Address: 11766

```
property ain33_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN33_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@33)

Address: 12066

```
property ain33_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN33_EF_INDEX (AIN#(0:149)_EF_INDEX@33)

Address: 9066

```
property ain33_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN33_EF_READ_A (AIN#(0:149)_EF_READ_A@33)

Address: 7066

```
property ain33_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN33_EF_READ_B (AIN#(0:149)_EF_READ_B@33)

Address: 7366

```
property ain33_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN33_EF_READ_C (AIN#(0:149)_EF_READ_C@33)

Address: 7666

```
property ain33_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN33_EF_READ_D (AIN#(0:149)_EF_READ_D@33)

Address: 7966

```
property ain34_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@34)

Address: 9368

```
property ain34_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@34)

Address: 9668

```
property ain34_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@34)

Address: 9968

```
property ain34_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@34)

Address: 10268

```
property ain34_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@34)

Address: 10568

```
property ain34_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@34)

Address: 10868

```
property ain34_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@34)

Address: 11168

```
property ain34_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@34)

Address: 11468

```
property ain34_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@34)

Address: 11768

```
property ain34_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN34_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@34)

Address: 12068

```
property ain34_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN34_EF_INDEX (AIN#(0:149)_EF_INDEX@34)

Address: 9068

```
property ain34_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN34_EF_READ_A (AIN#(0:149)_EF_READ_A@34)

Address: 7068

```
property ain34_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN34_EF_READ_B (AIN#(0:149)_EF_READ_B@34)

Address: 7368

```
property ain34_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN34_EF_READ_C (AIN#(0:149)_EF_READ_C@34)

Address: 7668

```
property ain34_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN34_EF_READ_D (AIN#(0:149)_EF_READ_D@34)

Address: 7968

```
property ain35_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@35)

Address: 9370

```
property ain35_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@35)

Address: 9670

```
property ain35_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@35)

Address: 9970

```
property ain35_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@35)

Address: 10270

```
property ain35_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@35)

Address: 10570

```
property ain35_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@35)

Address: 10870

```
property ain35_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@35)

Address: 11170

```
property ain35_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@35)

Address: 11470

```
property ain35_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@35)

Address: 11770

```
property ain35_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN35_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@35)

Address: 12070

```
property ain35_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN35_EF_INDEX (AIN#(0:149)_EF_INDEX@35)

Address: 9070

```
property ain35_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN35_EF_READ_A (AIN#(0:149)_EF_READ_A@35)

Address: 7070

```
property ain35_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN35_EF_READ_B (AIN#(0:149)_EF_READ_B@35)

Address: 7370

```
property ain35_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN35_EF_READ_C (AIN#(0:149)_EF_READ_C@35)

Address: 7670

```
property ain35_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN35_EF_READ_D (AIN#(0:149)_EF_READ_D@35)

Address: 7970

```
property ain36_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@36)

Address: 9372

```
property ain36_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@36)

Address: 9672

```
property ain36_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@36)

Address: 9972

```
property ain36_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@36)

Address: 10272

```
property ain36_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@36)

Address: 10572

```
property ain36_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@36)

Address: 10872

```
property ain36_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@36)

Address: 11172

```
property ain36_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@36)

Address: 11472

```
property ain36_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@36)

Address: 11772

```
property ain36_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN36_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@36)

Address: 12072

```
property ain36_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN36_EF_INDEX (AIN#(0:149)_EF_INDEX@36)

Address: 9072

```
property ain36_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN36_EF_READ_A (AIN#(0:149)_EF_READ_A@36)

Address: 7072

```
property ain36_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN36_EF_READ_B (AIN#(0:149)_EF_READ_B@36)

Address: 7372

```
property ain36_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN36_EF_READ_C (AIN#(0:149)_EF_READ_C@36)

Address: 7672

```
property ain36_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN36_EF_READ_D (AIN#(0:149)_EF_READ_D@36)

Address: 7972

```
property ain37_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@37)

Address: 9374

```
property ain37_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@37)

Address: 9674

```
property ain37_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@37)

Address: 9974

```
property ain37_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@37)

Address: 10274

```
property ain37_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@37)

Address: 10574

```
property ain37_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@37)

Address: 10874

```
property ain37_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@37)

Address: 11174

```
property ain37_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@37)

Address: 11474

```
property ain37_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@37)

Address: 11774

```
property ain37_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN37_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@37)

Address: 12074

```
property ain37_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN37_EF_INDEX (AIN#(0:149)_EF_INDEX@37)

Address: 9074

```
property ain37_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN37_EF_READ_A (AIN#(0:149)_EF_READ_A@37)

Address: 7074

```
property ain37_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN37_EF_READ_B (AIN#(0:149)_EF_READ_B@37)

Address: 7374

```
property ain37_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN37_EF_READ_C (AIN#(0:149)_EF_READ_C@37)

Address: 7674

```
property ain37_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN37_EF_READ_D (AIN#(0:149)_EF_READ_D@37)

Address: 7974

```
property ain38_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@38)

Address: 9376

```
property ain38_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@38)

Address: 9676

```
property ain38_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@38)

Address: 9976

```
property ain38_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@38)

Address: 10276

```
property ain38_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@38)

Address: 10576

```
property ain38_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@38)

Address: 10876

```
property ain38_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@38)

Address: 11176

```
property ain38_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@38)

Address: 11476

```
property ain38_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@38)

Address: 11776

```
property ain38_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN38_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@38)

Address: 12076

```
property ain38_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN38_EF_INDEX (AIN#(0:149)_EF_INDEX@38)

Address: 9076

```
property ain38_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN38_EF_READ_A (AIN#(0:149)_EF_READ_A@38)

Address: 7076

```
property ain38_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN38_EF_READ_B (AIN#(0:149)_EF_READ_B@38)

Address: 7376

```
property ain38_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN38_EF_READ_C (AIN#(0:149)_EF_READ_C@38)

Address: 7676

```
property ain38_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN38_EF_READ_D (AIN#(0:149)_EF_READ_D@38)

Address: 7976

```
property ain39_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@39)

Address: 9378

```
property ain39_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@39)

Address: 9678

```
property ain39_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@39)

Address: 9978

```
property ain39_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@39)

Address: 10278

```
property ain39_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@39)

Address: 10578

```
property ain39_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@39)

Address: 10878

```
property ain39_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@39)

Address: 11178

```
property ain39_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@39)

Address: 11478

```
property ain39_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@39)

Address: 11778

```
property ain39_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN39_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@39)

Address: 12078

```
property ain39_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN39_EF_INDEX (AIN#(0:149)_EF_INDEX@39)

Address: 9078

```
property ain39_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN39_EF_READ_A (AIN#(0:149)_EF_READ_A@39)

Address: 7078

```
property ain39_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN39_EF_READ_B (AIN#(0:149)_EF_READ_B@39)

Address: 7378

```
property ain39_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN39_EF_READ_C (AIN#(0:149)_EF_READ_C@39)

Address: 7678

```
property ain39_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN39_EF_READ_D (AIN#(0:149)_EF_READ_D@39)

Address: 7978

```
property ain3_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN3_BINARY (AIN#(0:7)_BINARY@3)

Address: 50006

```
property ain3_binary_capture: int
```

T8 Only. Returns the saved 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN3_BINARY_CAPTURE (AIN#(0:7)_BINARY_CAPTURE@3)

Address: 50656

```
property ain3_capture: float
```

T8 Only. Returns the saved voltage of the specified analog input.

Register: AIN3_CAPTURE (AIN#(0:7)_CAPTURE@3)

Address: 656

```
property ain3_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@3)

Address: 9306

```
property ain3_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@3)

Address: 9606

```
property ain3_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@3)

Address: 9906

```
property ain3_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@3)

Address: 10206

```
property ain3_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@3)

Address: 10506

```
property ain3_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@3)

Address: 10806

```
property ain3_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@3)

Address: 11106

```
property ain3_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@3)

Address: 11406

```
property ain3_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@3)

Address: 11706

```
property ain3_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN3_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@3)

Address: 12006

```
property ain3_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN3_EF_INDEX (AIN#(0:149)_EF_INDEX@3)

Address: 9006

```
property ain3_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN3_EF_READ_A (AIN#(0:149)_EF_READ_A@3)

Address: 7006

```
property ain3_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN3_EF_READ_B (AIN#(0:149)_EF_READ_B@3)

Address: 7306

```
property ain3_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN3_EF_READ_C (AIN#(0:149)_EF_READ_C@3)

Address: 7606

```
property ain3_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN3_EF_READ_D (AIN#(0:149)_EF_READ_D@3)

Address: 7906

```
property ain3_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN3_NEGATIVE_CH (AIN#(0:7)_NEGATIVE_CH@3)

Address: 41003

```
property ain3_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN3_RANGE (AIN#(0:7)_RANGE@3)

Address: 40006

```
property ain3_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN3_RESOLUTION_INDEX (AIN#(0:7)_RESOLUTION_INDEX@3)

Address: 41503

```
property ain3_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN3_SETTLING_US (AIN#(0:7)_SETTLING_US@3)

Address: 42006

```
property ain4: float
```

Returns the voltage of the specified analog input.

Register: AIN4 (AIN#(0:7)@4)

Address: 8

```
property ain40_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@40)

Address: 9380

```
property ain40_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@40)

Address: 9680

```
property ain40_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@40)

Address: 9980

```
property ain40_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@40)

Address: 10280

```
property ain40_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@40)

Address: 10580

```
property ain40_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@40)

Address: 10880

```
property ain40_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@40)

Address: 11180

```
property ain40_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@40)

Address: 11480

```
property ain40_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@40)

Address: 11780

```
property ain40_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN40_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@40)

Address: 12080

```
property ain40_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN40_EF_INDEX (AIN#(0:149)_EF_INDEX@40)

Address: 9080

```
property ain40_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN40_EF_READ_A (AIN#(0:149)_EF_READ_A@40)

Address: 7080

```
property ain40_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN40_EF_READ_B (AIN#(0:149)_EF_READ_B@40)

Address: 7380

```
property ain40_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN40_EF_READ_C (AIN#(0:149)_EF_READ_C@40)

Address: 7680

```
property ain40_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN40_EF_READ_D (AIN#(0:149)_EF_READ_D@40)

Address: 7980

```
property ain41_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@41)

Address: 9382

```
property ain41_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@41)

Address: 9682

```
property ain41_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@41)

Address: 9982

```
property ain41_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@41)

Address: 10282

```
property ain41_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@41)

Address: 10582

```
property ain41_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@41)

Address: 10882

```
property ain41_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@41)

Address: 11182

```
property ain41_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@41)

Address: 11482

```
property ain41_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@41)

Address: 11782

```
property ain41_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN41_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@41)

Address: 12082

```
property ain41_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN41_EF_INDEX (AIN#(0:149)_EF_INDEX@41)

Address: 9082

```
property ain41_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN41_EF_READ_A (AIN#(0:149)_EF_READ_A@41)

Address: 7082

```
property ain41_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN41_EF_READ_B (AIN#(0:149)_EF_READ_B@41)

Address: 7382

```
property ain41_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN41_EF_READ_C (AIN#(0:149)_EF_READ_C@41)

Address: 7682

```
property ain41_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN41_EF_READ_D (AIN#(0:149)_EF_READ_D@41)

Address: 7982

```
property ain42_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@42)

Address: 9384

```
property ain42_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@42)

Address: 9684

```
property ain42_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@42)

Address: 9984

```
property ain42_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@42)

Address: 10284

```
property ain42_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@42)

Address: 10584

```
property ain42_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@42)

Address: 10884

```
property ain42_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@42)

Address: 11184

```
property ain42_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@42)

Address: 11484

```
property ain42_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@42)

Address: 11784

```
property ain42_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN42_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@42)

Address: 12084

```
property ain42_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN42_EF_INDEX (AIN#(0:149)_EF_INDEX@42)

Address: 9084

```
property ain42_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN42_EF_READ_A (AIN#(0:149)_EF_READ_A@42)

Address: 7084

```
property ain42_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN42_EF_READ_B (AIN#(0:149)_EF_READ_B@42)

Address: 7384

```
property ain42_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN42_EF_READ_C (AIN#(0:149)_EF_READ_C@42)

Address: 7684

```
property ain42_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN42_EF_READ_D (AIN#(0:149)_EF_READ_D@42)

Address: 7984

```
property ain43_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@43)

Address: 9386

```
property ain43_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@43)

Address: 9686

```
property ain43_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@43)

Address: 9986

```
property ain43_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@43)

Address: 10286

```
property ain43_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@43)

Address: 10586

```
property ain43_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@43)

Address: 10886

```
property ain43_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@43)

Address: 11186

```
property ain43_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@43)

Address: 11486

```
property ain43_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@43)

Address: 11786

```
property ain43_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN43_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@43)

Address: 12086

```
property ain43_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN43_EF_INDEX (AIN#(0:149)_EF_INDEX@43)

Address: 9086

```
property ain43_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN43_EF_READ_A (AIN#(0:149)_EF_READ_A@43)

Address: 7086

```
property ain43_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN43_EF_READ_B (AIN#(0:149)_EF_READ_B@43)

Address: 7386

```
property ain43_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN43_EF_READ_C (AIN#(0:149)_EF_READ_C@43)

Address: 7686

```
property ain43_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN43_EF_READ_D (AIN#(0:149)_EF_READ_D@43)

Address: 7986

```
property ain44_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@44)

Address: 9388

```
property ain44_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@44)

Address: 9688

```
property ain44_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@44)

Address: 9988

```
property ain44_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@44)

Address: 10288

```
property ain44_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@44)

Address: 10588

```
property ain44_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@44)

Address: 10888

```
property ain44_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@44)

Address: 11188

```
property ain44_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@44)

Address: 11488

```
property ain44_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@44)

Address: 11788

```
property ain44_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN44_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@44)

Address: 12088

```
property ain44_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN44_EF_INDEX (AIN#(0:149)_EF_INDEX@44)

Address: 9088

```
property ain44_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN44_EF_READ_A (AIN#(0:149)_EF_READ_A@44)

Address: 7088

```
property ain44_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN44_EF_READ_B (AIN#(0:149)_EF_READ_B@44)

Address: 7388

```
property ain44_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN44_EF_READ_C (AIN#(0:149)_EF_READ_C@44)

Address: 7688

```
property ain44_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN44_EF_READ_D (AIN#(0:149)_EF_READ_D@44)

Address: 7988

```
property ain45_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@45)

Address: 9390

```
property ain45_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@45)

Address: 9690

```
property ain45_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@45)

Address: 9990

```
property ain45_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@45)

Address: 10290

```
property ain45_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@45)

Address: 10590

```
property ain45_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@45)

Address: 10890

```
property ain45_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@45)

Address: 11190

```
property ain45_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@45)

Address: 11490

```
property ain45_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@45)

Address: 11790

```
property ain45_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN45_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@45)

Address: 12090

```
property ain45_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN45_EF_INDEX (AIN#(0:149)_EF_INDEX@45)

Address: 9090

```
property ain45_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN45_EF_READ_A (AIN#(0:149)_EF_READ_A@45)

Address: 7090

```
property ain45_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN45_EF_READ_B (AIN#(0:149)_EF_READ_B@45)

Address: 7390

```
property ain45_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN45_EF_READ_C (AIN#(0:149)_EF_READ_C@45)

Address: 7690

```
property ain45_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN45_EF_READ_D (AIN#(0:149)_EF_READ_D@45)

Address: 7990

```
property ain46_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@46)

Address: 9392

```
property ain46_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@46)

Address: 9692

```
property ain46_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@46)

Address: 9992

```
property ain46_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@46)

Address: 10292

```
property ain46_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@46)

Address: 10592

```
property ain46_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@46)

Address: 10892

```
property ain46_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@46)

Address: 11192

```
property ain46_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@46)

Address: 11492

```
property ain46_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@46)

Address: 11792

```
property ain46_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN46_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@46)

Address: 12092

```
property ain46_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN46_EF_INDEX (AIN#(0:149)_EF_INDEX@46)

Address: 9092

```
property ain46_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN46_EF_READ_A (AIN#(0:149)_EF_READ_A@46)

Address: 7092

```
property ain46_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN46_EF_READ_B (AIN#(0:149)_EF_READ_B@46)

Address: 7392

```
property ain46_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN46_EF_READ_C (AIN#(0:149)_EF_READ_C@46)

Address: 7692

```
property ain46_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN46_EF_READ_D (AIN#(0:149)_EF_READ_D@46)

Address: 7992

```
property ain47_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@47)

Address: 9394

```
property ain47_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@47)

Address: 9694

```
property ain47_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@47)

Address: 9994

```
property ain47_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@47)

Address: 10294

```
property ain47_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@47)

Address: 10594

```
property ain47_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@47)

Address: 10894

```
property ain47_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@47)

Address: 11194

```
property ain47_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@47)

Address: 11494

```
property ain47_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@47)

Address: 11794

```
property ain47_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN47_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@47)

Address: 12094

```
property ain47_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN47_EF_INDEX (AIN#(0:149)_EF_INDEX@47)

Address: 9094

```
property ain47_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN47_EF_READ_A (AIN#(0:149)_EF_READ_A@47)

Address: 7094

```
property ain47_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN47_EF_READ_B (AIN#(0:149)_EF_READ_B@47)

Address: 7394

```
property ain47_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN47_EF_READ_C (AIN#(0:149)_EF_READ_C@47)

Address: 7694

```
property ain47_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN47_EF_READ_D (AIN#(0:149)_EF_READ_D@47)

Address: 7994

```
property ain48_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@48)

Address: 9396

```
property ain48_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@48)

Address: 9696

```
property ain48_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@48)

Address: 9996

```
property ain48_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@48)

Address: 10296

```
property ain48_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@48)

Address: 10596

```
property ain48_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@48)

Address: 10896

```
property ain48_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@48)

Address: 11196

```
property ain48_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@48)

Address: 11496

```
property ain48_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@48)

Address: 11796

```
property ain48_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN48_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@48)

Address: 12096

```
property ain48_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN48_EF_INDEX (AIN#(0:149)_EF_INDEX@48)

Address: 9096

```
property ain48_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN48_EF_READ_A (AIN#(0:149)_EF_READ_A@48)

Address: 7096

```
property ain48_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN48_EF_READ_B (AIN#(0:149)_EF_READ_B@48)

Address: 7396

```
property ain48_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN48_EF_READ_C (AIN#(0:149)_EF_READ_C@48)

Address: 7696

```
property ain48_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN48_EF_READ_D (AIN#(0:149)_EF_READ_D@48)

Address: 7996

```
property ain49_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@49)

Address: 9398

```
property ain49_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@49)

Address: 9698

```
property ain49_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@49)

Address: 9998

```
property ain49_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@49)

Address: 10298

```
property ain49_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@49)

Address: 10598

```
property ain49_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@49)

Address: 10898

```
property ain49_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@49)

Address: 11198

```
property ain49_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@49)

Address: 11498

```
property ain49_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@49)

Address: 11798

```
property ain49_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN49_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@49)

Address: 12098

```
property ain49_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN49_EF_INDEX (AIN#(0:149)_EF_INDEX@49)

Address: 9098

```
property ain49_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN49_EF_READ_A (AIN#(0:149)_EF_READ_A@49)

Address: 7098

```
property ain49_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN49_EF_READ_B (AIN#(0:149)_EF_READ_B@49)

Address: 7398

```
property ain49_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN49_EF_READ_C (AIN#(0:149)_EF_READ_C@49)

Address: 7698

```
property ain49_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN49_EF_READ_D (AIN#(0:149)_EF_READ_D@49)

Address: 7998

```
property ain4_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN4_BINARY (AIN#(0:7)_BINARY@4)

Address: 50008

```
property ain4_binary_capture: int
```

T8 Only. Returns the saved 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN4_BINARY_CAPTURE (AIN#(0:7)_BINARY_CAPTURE@4)

Address: 50658

```
property ain4_capture: float
```

T8 Only. Returns the saved voltage of the specified analog input.

Register: AIN4_CAPTURE (AIN#(0:7)_CAPTURE@4)

Address: 658

```
property ain4_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@4)

Address: 9308

```
property ain4_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@4)

Address: 9608

```
property ain4_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@4)

Address: 9908

```
property ain4_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@4)

Address: 10208

```
property ain4_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@4)

Address: 10508

```
property ain4_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@4)

Address: 10808

```
property ain4_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@4)

Address: 11108

```
property ain4_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@4)

Address: 11408

```
property ain4_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@4)

Address: 11708

```
property ain4_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN4_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@4)

Address: 12008

```
property ain4_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T;

25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN4_EF_INDEX (AIN#(0:149)_EF_INDEX@4)

Address: 9008

```
property ain4_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN4_EF_READ_A (AIN#(0:149)_EF_READ_A@4)

Address: 7008

```
property ain4_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN4_EF_READ_B (AIN#(0:149)_EF_READ_B@4)

Address: 7308

```
property ain4_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN4_EF_READ_C (AIN#(0:149)_EF_READ_C@4)

Address: 7608

```
property ain4_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN4_EF_READ_D (AIN#(0:149)_EF_READ_D@4)

Address: 7908

```
property ain4_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN4_NEGATIVE_CH (AIN#(0:7)_NEGATIVE_CH@4)

Address: 41004

```
property ain4_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN4_RANGE (AIN#(0:7)_RANGE@4)

Address: 40008

```
property ain4_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN4_RESOLUTION_INDEX (AIN#(0:7)_RESOLUTION_INDEX@4)

Address: 41504

```
property ain4_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN4_SETTLING_US (AIN#(0:7)_SETTLING_US@4)

Address: 42008

```
property ain5: float
```

Returns the voltage of the specified analog input.

Register: AIN5 (AIN#(0:7)@5)

Address: 10

```
property ain50_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@50)

Address: 9400

```
property ain50_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@50)

Address: 9700

```
property ain50_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@50)

Address: 10000

```
property ain50_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@50)

Address: 10300

```
property ain50_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@50)

Address: 10600

```
property ain50_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@50)

Address: 10900

```
property ain50_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@50)

Address: 11200

```
property ain50_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@50)

Address: 11500

```
property ain50_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@50)

Address: 11800

```
property ain50_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN50_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@50)

Address: 12100

```
property ain50_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN50_EF_INDEX (AIN#(0:149)_EF_INDEX@50)

Address: 9100

```
property ain50_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN50_EF_READ_A (AIN#(0:149)_EF_READ_A@50)

Address: 7100

```
property ain50_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN50_EF_READ_B (AIN#(0:149)_EF_READ_B@50)

Address: 7400

```
property ain50_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN50_EF_READ_C (AIN#(0:149)_EF_READ_C@50)

Address: 7700

```
property ain50_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN50_EF_READ_D (AIN#(0:149)_EF_READ_D@50)

Address: 8000

```
property ain51_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@51)

Address: 9402

```
property ain51_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@51)

Address: 9702

```
property ain51_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@51)

Address: 10002

```
property ain51_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@51)

Address: 10302

```
property ain51_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@51)

Address: 10602

```
property ain51_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@51)

Address: 10902

```
property ain51_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@51)

Address: 11202

```
property ain51_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@51)

Address: 11502

```
property ain51_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@51)

Address: 11802

```
property ain51_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN51_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@51)

Address: 12102

```
property ain51_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN51_EF_INDEX (AIN#(0:149)_EF_INDEX@51)

Address: 9102

```
property ain51_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN51_EF_READ_A (AIN#(0:149)_EF_READ_A@51)

Address: 7102

```
property ain51_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN51_EF_READ_B (AIN#(0:149)_EF_READ_B@51)

Address: 7402

```
property ain51_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN51_EF_READ_C (AIN#(0:149)_EF_READ_C@51)

Address: 7702

```
property ain51_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN51_EF_READ_D (AIN#(0:149)_EF_READ_D@51)

Address: 8002

```
property ain52_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@52)

Address: 9404

```
property ain52_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@52)

Address: 9704

```
property ain52_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@52)

Address: 10004

```
property ain52_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@52)

Address: 10304

```
property ain52_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@52)

Address: 10604

```
property ain52_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@52)

Address: 10904

```
property ain52_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@52)

Address: 11204

```
property ain52_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@52)

Address: 11504

```
property ain52_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@52)

Address: 11804

```
property ain52_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN52_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@52)

Address: 12104

```
property ain52_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN52_EF_INDEX (AIN#(0:149)_EF_INDEX@52)

Address: 9104

```
property ain52_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN52_EF_READ_A (AIN#(0:149)_EF_READ_A@52)

Address: 7104

```
property ain52_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN52_EF_READ_B (AIN#(0:149)_EF_READ_B@52)

Address: 7404

```
property ain52_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN52_EF_READ_C (AIN#(0:149)_EF_READ_C@52)

Address: 7704

```
property ain52_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN52_EF_READ_D (AIN#(0:149)_EF_READ_D@52)

Address: 8004

```
property ain53_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@53)

Address: 9406

```
property ain53_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@53)

Address: 9706

```
property ain53_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@53)

Address: 10006

```
property ain53_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@53)

Address: 10306

```
property ain53_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@53)

Address: 10606

```
property ain53_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@53)

Address: 10906

```
property ain53_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@53)

Address: 11206

```
property ain53_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@53)

Address: 11506

```
property ain53_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@53)

Address: 11806

```
property ain53_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN53_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@53)

Address: 12106

```
property ain53_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN53_EF_INDEX (AIN#(0:149)_EF_INDEX@53)

Address: 9106

```
property ain53_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN53_EF_READ_A (AIN#(0:149)_EF_READ_A@53)

Address: 7106

```
property ain53_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN53_EF_READ_B (AIN#(0:149)_EF_READ_B@53)

Address: 7406

```
property ain53_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN53_EF_READ_C (AIN#(0:149)_EF_READ_C@53)

Address: 7706

```
property ain53_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN53_EF_READ_D (AIN#(0:149)_EF_READ_D@53)

Address: 8006

```
property ain54_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@54)

Address: 9408

```
property ain54_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@54)

Address: 9708

```
property ain54_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@54)

Address: 10008

```
property ain54_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@54)

Address: 10308

```
property ain54_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@54)

Address: 10608

```
property ain54_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@54)

Address: 10908

```
property ain54_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@54)

Address: 11208

```
property ain54_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@54)

Address: 11508

```
property ain54_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@54)

Address: 11808

```
property ain54_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN54_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@54)

Address: 12108

```
property ain54_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN54_EF_INDEX (AIN#(0:149)_EF_INDEX@54)

Address: 9108

```
property ain54_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN54_EF_READ_A (AIN#(0:149)_EF_READ_A@54)

Address: 7108

```
property ain54_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN54_EF_READ_B (AIN#(0:149)_EF_READ_B@54)

Address: 7408

```
property ain54_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN54_EF_READ_C (AIN#(0:149)_EF_READ_C@54)

Address: 7708

```
property ain54_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN54_EF_READ_D (AIN#(0:149)_EF_READ_D@54)

Address: 8008

```
property ain55_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@55)

Address: 9410

```
property ain55_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@55)

Address: 9710

```
property ain55_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@55)

Address: 10010

```
property ain55_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@55)

Address: 10310

```
property ain55_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@55)

Address: 10610

```
property ain55_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@55)

Address: 10910

```
property ain55_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@55)

Address: 11210

```
property ain55_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@55)

Address: 11510

```
property ain55_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@55)

Address: 11810

```
property ain55_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN55_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@55)

Address: 12110

```
property ain55_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN55_EF_INDEX (AIN#(0:149)_EF_INDEX@55)

Address: 9110

```
property ain55_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN55_EF_READ_A (AIN#(0:149)_EF_READ_A@55)

Address: 7110

```
property ain55_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN55_EF_READ_B (AIN#(0:149)_EF_READ_B@55)

Address: 7410

```
property ain55_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN55_EF_READ_C (AIN#(0:149)_EF_READ_C@55)

Address: 7710

```
property ain55_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN55_EF_READ_D (AIN#(0:149)_EF_READ_D@55)

Address: 8010

```
property ain56_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@56)

Address: 9412

```
property ain56_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@56)

Address: 9712

```
property ain56_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@56)

Address: 10012

```
property ain56_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@56)

Address: 10312

```
property ain56_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@56)

Address: 10612

```
property ain56_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@56)

Address: 10912

```
property ain56_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@56)

Address: 11212

```
property ain56_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@56)

Address: 11512

```
property ain56_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@56)

Address: 11812

```
property ain56_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN56_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@56)

Address: 12112

```
property ain56_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN56_EF_INDEX (AIN#(0:149)_EF_INDEX@56)

Address: 9112

```
property ain56_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN56_EF_READ_A (AIN#(0:149)_EF_READ_A@56)

Address: 7112

```
property ain56_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN56_EF_READ_B (AIN#(0:149)_EF_READ_B@56)

Address: 7412

```
property ain56_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN56_EF_READ_C (AIN#(0:149)_EF_READ_C@56)

Address: 7712

```
property ain56_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN56_EF_READ_D (AIN#(0:149)_EF_READ_D@56)

Address: 8012

```
property ain57_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@57)

Address: 9414

```
property ain57_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@57)

Address: 9714

```
property ain57_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@57)

Address: 10014

```
property ain57_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@57)

Address: 10314

```
property ain57_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@57)

Address: 10614

```
property ain57_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@57)

Address: 10914

```
property ain57_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@57)

Address: 11214

```
property ain57_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@57)

Address: 11514

```
property ain57_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@57)

Address: 11814

```
property ain57_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN57_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@57)

Address: 12114

```
property ain57_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN57_EF_INDEX (AIN#(0:149)_EF_INDEX@57)

Address: 9114

```
property ain57_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN57_EF_READ_A (AIN#(0:149)_EF_READ_A@57)

Address: 7114

```
property ain57_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN57_EF_READ_B (AIN#(0:149)_EF_READ_B@57)

Address: 7414

```
property ain57_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN57_EF_READ_C (AIN#(0:149)_EF_READ_C@57)

Address: 7714

```
property ain57_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN57_EF_READ_D (AIN#(0:149)_EF_READ_D@57)

Address: 8014

```
property ain58_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@58)

Address: 9416

```
property ain58_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@58)

Address: 9716

```
property ain58_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@58)

Address: 10016

```
property ain58_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@58)

Address: 10316

```
property ain58_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@58)

Address: 10616

```
property ain58_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@58)

Address: 10916

```
property ain58_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@58)

Address: 11216

```
property ain58_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@58)

Address: 11516

```
property ain58_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@58)

Address: 11816

```
property ain58_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN58_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@58)

Address: 12116

```
property ain58_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN58_EF_INDEX (AIN#(0:149)_EF_INDEX@58)

Address: 9116

```
property ain58_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN58_EF_READ_A (AIN#(0:149)_EF_READ_A@58)

Address: 7116

```
property ain58_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN58_EF_READ_B (AIN#(0:149)_EF_READ_B@58)

Address: 7416

```
property ain58_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN58_EF_READ_C (AIN#(0:149)_EF_READ_C@58)

Address: 7716

```
property ain58_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN58_EF_READ_D (AIN#(0:149)_EF_READ_D@58)

Address: 8016

```
property ain59_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@59)

Address: 9418

```
property ain59_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@59)

Address: 9718

```
property ain59_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@59)

Address: 10018

```
property ain59_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@59)

Address: 10318

```
property ain59_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@59)

Address: 10618

```
property ain59_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@59)

Address: 10918

```
property ain59_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@59)

Address: 11218

```
property ain59_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@59)

Address: 11518

```
property ain59_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@59)

Address: 11818

```
property ain59_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN59_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@59)

Address: 12118

```
property ain59_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN59_EF_INDEX (AIN#(0:149)_EF_INDEX@59)

Address: 9118

```
property ain59_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN59_EF_READ_A (AIN#(0:149)_EF_READ_A@59)

Address: 7118

```
property ain59_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN59_EF_READ_B (AIN#(0:149)_EF_READ_B@59)

Address: 7418

```
property ain59_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN59_EF_READ_C (AIN#(0:149)_EF_READ_C@59)

Address: 7718

```
property ain59_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN59_EF_READ_D (AIN#(0:149)_EF_READ_D@59)

Address: 8018

```
property ain5_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN5_BINARY (AIN#(0:7)_BINARY@5)

Address: 50010

```
property ain5_binary_capture: int
```

T8 Only. Returns the saved 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN5_BINARY_CAPTURE (AIN#(0:7)_BINARY_CAPTURE@5)

Address: 50660

```
property ain5_capture: float
```

T8 Only. Returns the saved voltage of the specified analog input.

Register: AIN5_CAPTURE (AIN#(0:7)_CAPTURE@5)

Address: 660

```
property ain5_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@5)

Address: 9310

```
property ain5_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@5)

Address: 9610

```
property ain5_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@5)

Address: 9910

```
property ain5_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@5)

Address: 10210

```
property ain5_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@5)

Address: 10510

```
property ain5_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@5)

Address: 10810

```
property ain5_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@5)

Address: 11110

```
property ain5_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@5)

Address: 11410

```
property ain5_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@5)

Address: 11710

```
property ain5_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN5_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@5)

Address: 12010

```
property ain5_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN5_EF_INDEX (AIN#(0:149)_EF_INDEX@5)

Address: 9010

```
property ain5_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN5_EF_READ_A (AIN#(0:149)_EF_READ_A@5)

Address: 7010

```
property ain5_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN5_EF_READ_B (AIN#(0:149)_EF_READ_B@5)

Address: 7310

```
property ain5_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN5_EF_READ_C (AIN#(0:149)_EF_READ_C@5)

Address: 7610

```
property ain5_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN5_EF_READ_D (AIN#(0:149)_EF_READ_D@5)

Address: 7910

```
property ain5_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN5_NEGATIVE_CH (AIN#(0:7)_NEGATIVE_CH@5)

Address: 41005

```
property ain5_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN5_RANGE (AIN#(0:7)_RANGE@5)

Address: 40010

```
property ain5_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN5_RESOLUTION_INDEX (AIN#(0:7)_RESOLUTION_INDEX@5)

Address: 41505

```
property ain5_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN5_SETTLING_US (AIN#(0:7)_SETTLING_US@5)

Address: 42010

```
property ain6: float
```

Returns the voltage of the specified analog input.

Register: AIN6 (AIN#(0:7)@6)

Address: 12

```
property ain60_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@60)

Address: 9420

```
property ain60_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@60)

Address: 9720

```
property ain60_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@60)

Address: 10020

```
property ain60_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@60)

Address: 10320

```
property ain60_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@60)

Address: 10620

```
property ain60_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@60)

Address: 10920

```
property ain60_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@60)

Address: 11220

```
property ain60_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@60)

Address: 11520

```
property ain60_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@60)

Address: 11820

```
property ain60_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN60_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@60)

Address: 12120

```
property ain60_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN60_EF_INDEX (AIN#(0:149)_EF_INDEX@60)

Address: 9120

```
property ain60_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN60_EF_READ_A (AIN#(0:149)_EF_READ_A@60)

Address: 7120

```
property ain60_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN60_EF_READ_B (AIN#(0:149)_EF_READ_B@60)

Address: 7420

```
property ain60_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN60_EF_READ_C (AIN#(0:149)_EF_READ_C@60)

Address: 7720

```
property ain60_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN60_EF_READ_D (AIN#(0:149)_EF_READ_D@60)

Address: 8020

```
property ain61_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@61)

Address: 9422

```
property ain61_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@61)

Address: 9722

```
property ain61_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@61)

Address: 10022

```
property ain61_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@61)

Address: 10322

```
property ain61_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@61)

Address: 10622

```
property ain61_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@61)

Address: 10922

```
property ain61_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@61)

Address: 11222

```
property ain61_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@61)

Address: 11522

```
property ain61_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@61)

Address: 11822

```
property ain61_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN61_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@61)

Address: 12122

```
property ain61_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN61_EF_INDEX (AIN#(0:149)_EF_INDEX@61)

Address: 9122

```
property ain61_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN61_EF_READ_A (AIN#(0:149)_EF_READ_A@61)

Address: 7122

```
property ain61_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN61_EF_READ_B (AIN#(0:149)_EF_READ_B@61)

Address: 7422

```
property ain61_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN61_EF_READ_C (AIN#(0:149)_EF_READ_C@61)

Address: 7722

```
property ain61_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN61_EF_READ_D (AIN#(0:149)_EF_READ_D@61)

Address: 8022

```
property ain62_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@62)

Address: 9424

```
property ain62_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@62)

Address: 9724

```
property ain62_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@62)

Address: 10024

```
property ain62_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@62)

Address: 10324

```
property ain62_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@62)

Address: 10624

```
property ain62_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@62)

Address: 10924

```
property ain62_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@62)

Address: 11224

```
property ain62_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@62)

Address: 11524

```
property ain62_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@62)

Address: 11824

```
property ain62_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN62_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@62)

Address: 12124

```
property ain62_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN62_EF_INDEX (AIN#(0:149)_EF_INDEX@62)

Address: 9124

```
property ain62_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN62_EF_READ_A (AIN#(0:149)_EF_READ_A@62)

Address: 7124

```
property ain62_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN62_EF_READ_B (AIN#(0:149)_EF_READ_B@62)

Address: 7424

```
property ain62_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN62_EF_READ_C (AIN#(0:149)_EF_READ_C@62)

Address: 7724

```
property ain62_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN62_EF_READ_D (AIN#(0:149)_EF_READ_D@62)

Address: 8024

```
property ain63_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@63)

Address: 9426

```
property ain63_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@63)

Address: 9726

```
property ain63_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@63)

Address: 10026

```
property ain63_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@63)

Address: 10326

```
property ain63_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@63)

Address: 10626

```
property ain63_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@63)

Address: 10926

```
property ain63_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@63)

Address: 11226

```
property ain63_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@63)

Address: 11526

```
property ain63_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@63)

Address: 11826

```
property ain63_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN63_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@63)

Address: 12126

```
property ain63_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN63_EF_INDEX (AIN#(0:149)_EF_INDEX@63)

Address: 9126

```
property ain63_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN63_EF_READ_A (AIN#(0:149)_EF_READ_A@63)

Address: 7126

```
property ain63_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN63_EF_READ_B (AIN#(0:149)_EF_READ_B@63)

Address: 7426

```
property ain63_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN63_EF_READ_C (AIN#(0:149)_EF_READ_C@63)

Address: 7726

```
property ain63_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN63_EF_READ_D (AIN#(0:149)_EF_READ_D@63)

Address: 8026

```
property ain64_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@64)

Address: 9428

```
property ain64_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@64)

Address: 9728

```
property ain64_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@64)

Address: 10028

```
property ain64_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@64)

Address: 10328

```
property ain64_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@64)

Address: 10628

```
property ain64_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@64)

Address: 10928

```
property ain64_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@64)

Address: 11228

```
property ain64_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@64)

Address: 11528

```
property ain64_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@64)

Address: 11828

```
property ain64_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN64_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@64)

Address: 12128

```
property ain64_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN64_EF_INDEX (AIN#(0:149)_EF_INDEX@64)

Address: 9128

```
property ain64_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN64_EF_READ_A (AIN#(0:149)_EF_READ_A@64)

Address: 7128

```
property ain64_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN64_EF_READ_B (AIN#(0:149)_EF_READ_B@64)

Address: 7428

```
property ain64_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN64_EF_READ_C (AIN#(0:149)_EF_READ_C@64)

Address: 7728

```
property ain64_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN64_EF_READ_D (AIN#(0:149)_EF_READ_D@64)

Address: 8028

```
property ain65_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@65)

Address: 9430

```
property ain65_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@65)

Address: 9730

```
property ain65_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@65)

Address: 10030

```
property ain65_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@65)

Address: 10330

```
property ain65_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@65)

Address: 10630

```
property ain65_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@65)

Address: 10930

```
property ain65_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@65)

Address: 11230

```
property ain65_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@65)

Address: 11530

```
property ain65_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@65)

Address: 11830

```
property ain65_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN65_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@65)

Address: 12130

```
property ain65_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN65_EF_INDEX (AIN#(0:149)_EF_INDEX@65)

Address: 9130

```
property ain65_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN65_EF_READ_A (AIN#(0:149)_EF_READ_A@65)

Address: 7130

```
property ain65_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN65_EF_READ_B (AIN#(0:149)_EF_READ_B@65)

Address: 7430

```
property ain65_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN65_EF_READ_C (AIN#(0:149)_EF_READ_C@65)

Address: 7730

```
property ain65_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN65_EF_READ_D (AIN#(0:149)_EF_READ_D@65)

Address: 8030

```
property ain66_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@66)

Address: 9432

```
property ain66_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@66)

Address: 9732

```
property ain66_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@66)

Address: 10032

```
property ain66_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@66)

Address: 10332

```
property ain66_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@66)

Address: 10632

```
property ain66_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@66)

Address: 10932

```
property ain66_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@66)

Address: 11232

```
property ain66_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@66)

Address: 11532

```
property ain66_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@66)

Address: 11832

```
property ain66_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN66_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@66)

Address: 12132

```
property ain66_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN66_EF_INDEX (AIN#(0:149)_EF_INDEX@66)

Address: 9132

```
property ain66_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN66_EF_READ_A (AIN#(0:149)_EF_READ_A@66)

Address: 7132

```
property ain66_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN66_EF_READ_B (AIN#(0:149)_EF_READ_B@66)

Address: 7432

```
property ain66_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN66_EF_READ_C (AIN#(0:149)_EF_READ_C@66)

Address: 7732

```
property ain66_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN66_EF_READ_D (AIN#(0:149)_EF_READ_D@66)

Address: 8032

```
property ain67_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@67)

Address: 9434

```
property ain67_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@67)

Address: 9734

```
property ain67_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@67)

Address: 10034

```
property ain67_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@67)

Address: 10334

```
property ain67_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@67)

Address: 10634

```
property ain67_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@67)

Address: 10934

```
property ain67_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@67)

Address: 11234

```
property ain67_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@67)

Address: 11534

```
property ain67_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@67)

Address: 11834

```
property ain67_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN67_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@67)

Address: 12134

```
property ain67_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN67_EF_INDEX (AIN#(0:149)_EF_INDEX@67)

Address: 9134

```
property ain67_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN67_EF_READ_A (AIN#(0:149)_EF_READ_A@67)

Address: 7134

```
property ain67_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN67_EF_READ_B (AIN#(0:149)_EF_READ_B@67)

Address: 7434

```
property ain67_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN67_EF_READ_C (AIN#(0:149)_EF_READ_C@67)

Address: 7734

```
property ain67_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN67_EF_READ_D (AIN#(0:149)_EF_READ_D@67)

Address: 8034

```
property ain68_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@68)

Address: 9436

```
property ain68_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@68)

Address: 9736

```
property ain68_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@68)

Address: 10036

```
property ain68_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@68)

Address: 10336

```
property ain68_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@68)

Address: 10636

```
property ain68_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@68)

Address: 10936

```
property ain68_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@68)

Address: 11236

```
property ain68_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@68)

Address: 11536

```
property ain68_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@68)

Address: 11836

```
property ain68_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN68_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@68)

Address: 12136

```
property ain68_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN68_EF_INDEX (AIN#(0:149)_EF_INDEX@68)

Address: 9136

```
property ain68_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN68_EF_READ_A (AIN#(0:149)_EF_READ_A@68)

Address: 7136

```
property ain68_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN68_EF_READ_B (AIN#(0:149)_EF_READ_B@68)

Address: 7436

```
property ain68_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN68_EF_READ_C (AIN#(0:149)_EF_READ_C@68)

Address: 7736

```
property ain68_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN68_EF_READ_D (AIN#(0:149)_EF_READ_D@68)

Address: 8036

```
property ain69_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@69)

Address: 9438

```
property ain69_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@69)

Address: 9738

```
property ain69_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@69)

Address: 10038

```
property ain69_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@69)

Address: 10338

```
property ain69_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@69)

Address: 10638

```
property ain69_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@69)

Address: 10938

```
property ain69_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@69)

Address: 11238

```
property ain69_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@69)

Address: 11538

```
property ain69_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@69)

Address: 11838

```
property ain69_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN69_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@69)

Address: 12138

```
property ain69_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN69_EF_INDEX (AIN#(0:149)_EF_INDEX@69)

Address: 9138

```
property ain69_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN69_EF_READ_A (AIN#(0:149)_EF_READ_A@69)

Address: 7138

```
property ain69_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN69_EF_READ_B (AIN#(0:149)_EF_READ_B@69)

Address: 7438

```
property ain69_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN69_EF_READ_C (AIN#(0:149)_EF_READ_C@69)

Address: 7738

```
property ain69_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN69_EF_READ_D (AIN#(0:149)_EF_READ_D@69)

Address: 8038

```
property ain6_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN6_BINARY (AIN#(0:7)_BINARY@6)

Address: 50012

```
property ain6_binary_capture: int
```

T8 Only. Returns the saved 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN6_BINARY_CAPTURE (AIN#(0:7)_BINARY_CAPTURE@6)

Address: 50662

```
property ain6_capture: float
```

T8 Only. Returns the saved voltage of the specified analog input.

Register: AIN6_CAPTURE (AIN#(0:7)_CAPTURE@6)

Address: 662

```
property ain6_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@6)

Address: 9312

```
property ain6_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@6)

Address: 9612

```
property ain6_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@6)

Address: 9912

```
property ain6_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@6)

Address: 10212

```
property ain6_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@6)

Address: 10512

```
property ain6_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@6)

Address: 10812

```
property ain6_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@6)

Address: 11112

```
property ain6_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@6)

Address: 11412

```
property ain6_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@6)

Address: 11712

```
property ain6_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN6_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@6)

Address: 12012

```
property ain6_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN6_EF_INDEX (AIN#(0:149)_EF_INDEX@6)

Address: 9012

```
property ain6_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN6_EF_READ_A (AIN#(0:149)_EF_READ_A@6)

Address: 7012

```
property ain6_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN6_EF_READ_B (AIN#(0:149)_EF_READ_B@6)

Address: 7312

```
property ain6_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN6_EF_READ_C (AIN#(0:149)_EF_READ_C@6)

Address: 7612

```
property ain6_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN6_EF_READ_D (AIN#(0:149)_EF_READ_D@6)

Address: 7912

```
property ain6_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN6_NEGATIVE_CH (AIN#(0:7)_NEGATIVE_CH@6)

Address: 41006

```
property ain6_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN6_RANGE (AIN#(0:7)_RANGE@6)

Address: 40012

```
property ain6_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN6_RESOLUTION_INDEX (AIN#(0:7)_RESOLUTION_INDEX@6)

Address: 41506

```
property ain6_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN6_SETTLING_US (AIN#(0:7)_SETTLING_US@6)

Address: 42012

```
property ain7: float
```

Returns the voltage of the specified analog input.

Register: AIN7 (AIN#(0:7)@7)

Address: 14

```
property ain70_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@70)

Address: 9440

```
property ain70_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@70)

Address: 9740

```
property ain70_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@70)

Address: 10040

```
property ain70_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@70)

Address: 10340

```
property ain70_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@70)

Address: 10640

```
property ain70_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@70)

Address: 10940

```
property ain70_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@70)

Address: 11240

```
property ain70_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@70)

Address: 11540

```
property ain70_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@70)

Address: 11840

```
property ain70_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN70_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@70)

Address: 12140

```
property ain70_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN70_EF_INDEX (AIN#(0:149)_EF_INDEX@70)

Address: 9140

```
property ain70_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN70_EF_READ_A (AIN#(0:149)_EF_READ_A@70)

Address: 7140

```
property ain70_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN70_EF_READ_B (AIN#(0:149)_EF_READ_B@70)

Address: 7440

```
property ain70_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN70_EF_READ_C (AIN#(0:149)_EF_READ_C@70)

Address: 7740

```
property ain70_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN70_EF_READ_D (AIN#(0:149)_EF_READ_D@70)

Address: 8040

```
property ain71_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@71)

Address: 9442

```
property ain71_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@71)

Address: 9742

```
property ain71_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@71)

Address: 10042

```
property ain71_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@71)

Address: 10342

```
property ain71_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@71)

Address: 10642

```
property ain71_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@71)

Address: 10942

```
property ain71_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@71)

Address: 11242

```
property ain71_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@71)

Address: 11542

```
property ain71_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@71)

Address: 11842

```
property ain71_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN71_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@71)

Address: 12142

```
property ain71_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN71_EF_INDEX (AIN#(0:149)_EF_INDEX@71)

Address: 9142

```
property ain71_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN71_EF_READ_A (AIN#(0:149)_EF_READ_A@71)

Address: 7142

```
property ain71_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN71_EF_READ_B (AIN#(0:149)_EF_READ_B@71)

Address: 7442

```
property ain71_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN71_EF_READ_C (AIN#(0:149)_EF_READ_C@71)

Address: 7742

```
property ain71_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN71_EF_READ_D (AIN#(0:149)_EF_READ_D@71)

Address: 8042

```
property ain72_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@72)

Address: 9444

```
property ain72_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@72)

Address: 9744

```
property ain72_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@72)

Address: 10044

```
property ain72_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@72)

Address: 10344

```
property ain72_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@72)

Address: 10644

```
property ain72_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@72)

Address: 10944

```
property ain72_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@72)

Address: 11244

```
property ain72_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@72)

Address: 11544

```
property ain72_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@72)

Address: 11844

```
property ain72_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN72_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@72)

Address: 12144

```
property ain72_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN72_EF_INDEX (AIN#(0:149)_EF_INDEX@72)

Address: 9144

```
property ain72_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN72_EF_READ_A (AIN#(0:149)_EF_READ_A@72)

Address: 7144

```
property ain72_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN72_EF_READ_B (AIN#(0:149)_EF_READ_B@72)

Address: 7444

```
property ain72_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN72_EF_READ_C (AIN#(0:149)_EF_READ_C@72)

Address: 7744

```
property ain72_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN72_EF_READ_D (AIN#(0:149)_EF_READ_D@72)

Address: 8044

```
property ain73_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@73)

Address: 9446

```
property ain73_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@73)

Address: 9746

```
property ain73_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@73)

Address: 10046

```
property ain73_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@73)

Address: 10346

```
property ain73_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@73)

Address: 10646

```
property ain73_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@73)

Address: 10946

```
property ain73_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@73)

Address: 11246

```
property ain73_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@73)

Address: 11546

```
property ain73_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@73)

Address: 11846

```
property ain73_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN73_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@73)

Address: 12146

```
property ain73_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN73_EF_INDEX (AIN#(0:149)_EF_INDEX@73)

Address: 9146

```
property ain73_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN73_EF_READ_A (AIN#(0:149)_EF_READ_A@73)

Address: 7146

```
property ain73_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN73_EF_READ_B (AIN#(0:149)_EF_READ_B@73)

Address: 7446

```
property ain73_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN73_EF_READ_C (AIN#(0:149)_EF_READ_C@73)

Address: 7746

```
property ain73_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN73_EF_READ_D (AIN#(0:149)_EF_READ_D@73)

Address: 8046

```
property ain74_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@74)

Address: 9448

```
property ain74_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@74)

Address: 9748

```
property ain74_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@74)

Address: 10048

```
property ain74_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@74)

Address: 10348

```
property ain74_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@74)

Address: 10648

```
property ain74_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@74)

Address: 10948

```
property ain74_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@74)

Address: 11248

```
property ain74_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@74)

Address: 11548

```
property ain74_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@74)

Address: 11848

```
property ain74_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN74_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@74)

Address: 12148

```
property ain74_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN74_EF_INDEX (AIN#(0:149)_EF_INDEX@74)

Address: 9148

```
property ain74_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN74_EF_READ_A (AIN#(0:149)_EF_READ_A@74)

Address: 7148

```
property ain74_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN74_EF_READ_B (AIN#(0:149)_EF_READ_B@74)

Address: 7448

```
property ain74_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN74_EF_READ_C (AIN#(0:149)_EF_READ_C@74)

Address: 7748

```
property ain74_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN74_EF_READ_D (AIN#(0:149)_EF_READ_D@74)

Address: 8048

```
property ain75_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@75)

Address: 9450

```
property ain75_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@75)

Address: 9750

```
property ain75_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@75)

Address: 10050

```
property ain75_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@75)

Address: 10350

```
property ain75_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@75)

Address: 10650

```
property ain75_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@75)

Address: 10950

```
property ain75_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@75)

Address: 11250

```
property ain75_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@75)

Address: 11550

```
property ain75_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@75)

Address: 11850

```
property ain75_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN75_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@75)

Address: 12150

```
property ain75_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN75_EF_INDEX (AIN#(0:149)_EF_INDEX@75)

Address: 9150

```
property ain75_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN75_EF_READ_A (AIN#(0:149)_EF_READ_A@75)

Address: 7150

```
property ain75_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN75_EF_READ_B (AIN#(0:149)_EF_READ_B@75)

Address: 7450

```
property ain75_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN75_EF_READ_C (AIN#(0:149)_EF_READ_C@75)

Address: 7750

```
property ain75_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN75_EF_READ_D (AIN#(0:149)_EF_READ_D@75)

Address: 8050

```
property ain76_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@76)

Address: 9452

```
property ain76_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@76)

Address: 9752

```
property ain76_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@76)

Address: 10052

```
property ain76_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@76)

Address: 10352

```
property ain76_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@76)

Address: 10652

```
property ain76_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@76)

Address: 10952

```
property ain76_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@76)

Address: 11252

```
property ain76_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@76)

Address: 11552

```
property ain76_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@76)

Address: 11852

```
property ain76_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN76_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@76)

Address: 12152

```
property ain76_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN76_EF_INDEX (AIN#(0:149)_EF_INDEX@76)

Address: 9152

```
property ain76_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN76_EF_READ_A (AIN#(0:149)_EF_READ_A@76)

Address: 7152

```
property ain76_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN76_EF_READ_B (AIN#(0:149)_EF_READ_B@76)

Address: 7452

```
property ain76_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN76_EF_READ_C (AIN#(0:149)_EF_READ_C@76)

Address: 7752

```
property ain76_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN76_EF_READ_D (AIN#(0:149)_EF_READ_D@76)

Address: 8052

```
property ain77_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@77)

Address: 9454

```
property ain77_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@77)

Address: 9754

```
property ain77_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@77)

Address: 10054

```
property ain77_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@77)

Address: 10354

```
property ain77_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@77)

Address: 10654

```
property ain77_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@77)

Address: 10954

```
property ain77_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@77)

Address: 11254

```
property ain77_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@77)

Address: 11554

```
property ain77_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@77)

Address: 11854

```
property ain77_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN77_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@77)

Address: 12154

```
property ain77_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN77_EF_INDEX (AIN#(0:149)_EF_INDEX@77)

Address: 9154

```
property ain77_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN77_EF_READ_A (AIN#(0:149)_EF_READ_A@77)

Address: 7154

```
property ain77_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN77_EF_READ_B (AIN#(0:149)_EF_READ_B@77)

Address: 7454

```
property ain77_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN77_EF_READ_C (AIN#(0:149)_EF_READ_C@77)

Address: 7754

```
property ain77_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN77_EF_READ_D (AIN#(0:149)_EF_READ_D@77)

Address: 8054

```
property ain78_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@78)

Address: 9456

```
property ain78_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@78)

Address: 9756

```
property ain78_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@78)

Address: 10056

```
property ain78_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@78)

Address: 10356

```
property ain78_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@78)

Address: 10656

```
property ain78_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@78)

Address: 10956

```
property ain78_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@78)

Address: 11256

```
property ain78_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@78)

Address: 11556

```
property ain78_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@78)

Address: 11856

```
property ain78_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN78_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@78)

Address: 12156

```
property ain78_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN78_EF_INDEX (AIN#(0:149)_EF_INDEX@78)

Address: 9156

```
property ain78_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN78_EF_READ_A (AIN#(0:149)_EF_READ_A@78)

Address: 7156

```
property ain78_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN78_EF_READ_B (AIN#(0:149)_EF_READ_B@78)

Address: 7456

```
property ain78_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN78_EF_READ_C (AIN#(0:149)_EF_READ_C@78)

Address: 7756

```
property ain78_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN78_EF_READ_D (AIN#(0:149)_EF_READ_D@78)

Address: 8056

```
property ain79_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@79)

Address: 9458

```
property ain79_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@79)

Address: 9758

```
property ain79_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@79)

Address: 10058

```
property ain79_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@79)

Address: 10358

```
property ain79_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@79)

Address: 10658

```
property ain79_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@79)

Address: 10958

```
property ain79_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@79)

Address: 11258

```
property ain79_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@79)

Address: 11558

```
property ain79_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@79)

Address: 11858

```
property ain79_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN79_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@79)

Address: 12158

```
property ain79_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN79_EF_INDEX (AIN#(0:149)_EF_INDEX@79)

Address: 9158

```
property ain79_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN79_EF_READ_A (AIN#(0:149)_EF_READ_A@79)

Address: 7158

```
property ain79_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN79_EF_READ_B (AIN#(0:149)_EF_READ_B@79)

Address: 7458

```
property ain79_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN79_EF_READ_C (AIN#(0:149)_EF_READ_C@79)

Address: 7758

```
property ain79_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN79_EF_READ_D (AIN#(0:149)_EF_READ_D@79)

Address: 8058

```
property ain7_binary: int
```

Returns the 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN7_BINARY (AIN#(0:7)_BINARY@7)

Address: 50014

```
property ain7_binary_capture: int
```

T8 Only. Returns the saved 24-bit binary representation of the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: AIN7_BINARY_CAPTURE (AIN#(0:7)_BINARY_CAPTURE@7)

Address: 50664

```
property ain7_capture: float
```

T8 Only. Returns the saved voltage of the specified analog input.

Register: AIN7_CAPTURE (AIN#(0:7)_CAPTURE@7)

Address: 664

```
property ain7_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@7)

Address: 9314

```
property ain7_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@7)

Address: 9614

```
property ain7_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@7)

Address: 9914

```
property ain7_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@7)

Address: 10214

```
property ain7_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@7)

Address: 10514

```
property ain7_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@7)

Address: 10814

```
property ain7_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@7)

Address: 11114

```
property ain7_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@7)

Address: 11414

```
property ain7_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@7)

Address: 11714

```
property ain7_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN7_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@7)

Address: 12014

```
property ain7_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN7_EF_INDEX (AIN#(0:149)_EF_INDEX@7)

Address: 9014

```
property ain7_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN7_EF_READ_A (AIN#(0:149)_EF_READ_A@7)

Address: 7014

```
property ain7_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN7_EF_READ_B (AIN#(0:149)_EF_READ_B@7)

Address: 7314

```
property ain7_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN7_EF_READ_C (AIN#(0:149)_EF_READ_C@7)

Address: 7614

```
property ain7_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN7_EF_READ_D (AIN#(0:149)_EF_READ_D@7)

Address: 7914

```
property ain7_negative_ch: int
```

Specifies the negative channel to be used for each positive channel. 199=Default=> Single-Ended.

Register: AIN7_NEGATIVE_CH (AIN#(0:7)_NEGATIVE_CH@7)

Address: 41007

```
property ain7_range: float
```

The range/span of each analog input. Write the highest expected input voltage.

Register: AIN7_RANGE (AIN#(0:7)_RANGE@7)

Address: 40014

```
property ain7_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times.

Register: AIN7_RESOLUTION_INDEX (AIN#(0:7)_RESOLUTION_INDEX@7)

Address: 41507

```
property ain7_settling_us: float
```

Settling time for command-response and AIN-EF readings.

Register: AIN7_SETTLING_US (AIN#(0:7)_SETTLING_US@7)

Address: 42014

```
property ain80_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@80)

Address: 9460

```
property ain80_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@80)

Address: 9760

```
property ain80_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@80)

Address: 10060

```
property ain80_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@80)

Address: 10360

```
property ain80_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@80)

Address: 10660

```
property ain80_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@80)

Address: 10960

```
property ain80_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@80)

Address: 11260

```
property ain80_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@80)

Address: 11560

```
property ain80_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@80)

Address: 11860

```
property ain80_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN80_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@80)

Address: 12160

```
property ain80_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN80_EF_INDEX (AIN#(0:149)_EF_INDEX@80)

Address: 9160

```
property ain80_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN80_EF_READ_A (AIN#(0:149)_EF_READ_A@80)

Address: 7160

```
property ain80_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN80_EF_READ_B (AIN#(0:149)_EF_READ_B@80)

Address: 7460

```
property ain80_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN80_EF_READ_C (AIN#(0:149)_EF_READ_C@80)

Address: 7760

```
property ain80_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN80_EF_READ_D (AIN#(0:149)_EF_READ_D@80)

Address: 8060

```
property ain81_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@81)

Address: 9462

```
property ain81_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@81)

Address: 9762

```
property ain81_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@81)

Address: 10062

```
property ain81_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@81)

Address: 10362

```
property ain81_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@81)

Address: 10662

```
property ain81_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@81)

Address: 10962

```
property ain81_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@81)

Address: 11262

```
property ain81_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@81)

Address: 11562

```
property ain81_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@81)

Address: 11862

```
property ain81_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN81_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@81)

Address: 12162

```
property ain81_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN81_EF_INDEX (AIN#(0:149)_EF_INDEX@81)

Address: 9162

```
property ain81_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN81_EF_READ_A (AIN#(0:149)_EF_READ_A@81)

Address: 7162

```
property ain81_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN81_EF_READ_B (AIN#(0:149)_EF_READ_B@81)

Address: 7462

```
property ain81_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN81_EF_READ_C (AIN#(0:149)_EF_READ_C@81)

Address: 7762

```
property ain81_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN81_EF_READ_D (AIN#(0:149)_EF_READ_D@81)

Address: 8062

```
property ain82_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@82)

Address: 9464

```
property ain82_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@82)

Address: 9764

```
property ain82_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@82)

Address: 10064

```
property ain82_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@82)

Address: 10364

```
property ain82_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@82)

Address: 10664

```
property ain82_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@82)

Address: 10964

```
property ain82_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@82)

Address: 11264

```
property ain82_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@82)

Address: 11564

```
property ain82_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@82)

Address: 11864

```
property ain82_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN82_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@82)

Address: 12164

```
property ain82_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN82_EF_INDEX (AIN#(0:149)_EF_INDEX@82)

Address: 9164

```
property ain82_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN82_EF_READ_A (AIN#(0:149)_EF_READ_A@82)

Address: 7164

```
property ain82_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN82_EF_READ_B (AIN#(0:149)_EF_READ_B@82)

Address: 7464

```
property ain82_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN82_EF_READ_C (AIN#(0:149)_EF_READ_C@82)

Address: 7764

```
property ain82_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN82_EF_READ_D (AIN#(0:149)_EF_READ_D@82)

Address: 8064

```
property ain83_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@83)

Address: 9466

```
property ain83_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@83)

Address: 9766

```
property ain83_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@83)

Address: 10066

```
property ain83_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@83)

Address: 10366

```
property ain83_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@83)

Address: 10666

```
property ain83_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@83)

Address: 10966

```
property ain83_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@83)

Address: 11266

```
property ain83_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@83)

Address: 11566

```
property ain83_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@83)

Address: 11866

```
property ain83_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN83_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@83)

Address: 12166

```
property ain83_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN83_EF_INDEX (AIN#(0:149)_EF_INDEX@83)

Address: 9166

```
property ain83_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN83_EF_READ_A (AIN#(0:149)_EF_READ_A@83)

Address: 7166

```
property ain83_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN83_EF_READ_B (AIN#(0:149)_EF_READ_B@83)

Address: 7466

```
property ain83_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN83_EF_READ_C (AIN#(0:149)_EF_READ_C@83)

Address: 7766

```
property ain83_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN83_EF_READ_D (AIN#(0:149)_EF_READ_D@83)

Address: 8066

```
property ain84_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@84)

Address: 9468

```
property ain84_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@84)

Address: 9768

```
property ain84_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@84)

Address: 10068

```
property ain84_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@84)

Address: 10368

```
property ain84_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@84)

Address: 10668

```
property ain84_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@84)

Address: 10968

```
property ain84_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@84)

Address: 11268

```
property ain84_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@84)

Address: 11568

```
property ain84_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@84)

Address: 11868

```
property ain84_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN84_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@84)

Address: 12168

```
property ain84_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN84_EF_INDEX (AIN#(0:149)_EF_INDEX@84)

Address: 9168

```
property ain84_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN84_EF_READ_A (AIN#(0:149)_EF_READ_A@84)

Address: 7168

```
property ain84_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN84_EF_READ_B (AIN#(0:149)_EF_READ_B@84)

Address: 7468

```
property ain84_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN84_EF_READ_C (AIN#(0:149)_EF_READ_C@84)

Address: 7768

```
property ain84_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN84_EF_READ_D (AIN#(0:149)_EF_READ_D@84)

Address: 8068

```
property ain85_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@85)

Address: 9470

```
property ain85_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@85)

Address: 9770

```
property ain85_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@85)

Address: 10070

```
property ain85_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@85)

Address: 10370

```
property ain85_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@85)

Address: 10670

```
property ain85_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@85)

Address: 10970

```
property ain85_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@85)

Address: 11270

```
property ain85_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@85)

Address: 11570

```
property ain85_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@85)

Address: 11870

```
property ain85_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN85_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@85)

Address: 12170

```
property ain85_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN85_EF_INDEX (AIN#(0:149)_EF_INDEX@85)

Address: 9170

```
property ain85_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN85_EF_READ_A (AIN#(0:149)_EF_READ_A@85)

Address: 7170

```
property ain85_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN85_EF_READ_B (AIN#(0:149)_EF_READ_B@85)

Address: 7470

```
property ain85_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN85_EF_READ_C (AIN#(0:149)_EF_READ_C@85)

Address: 7770

```
property ain85_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN85_EF_READ_D (AIN#(0:149)_EF_READ_D@85)

Address: 8070

```
property ain86_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@86)

Address: 9472

```
property ain86_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@86)

Address: 9772

```
property ain86_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@86)

Address: 10072

```
property ain86_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@86)

Address: 10372

```
property ain86_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@86)

Address: 10672

```
property ain86_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@86)

Address: 10972

```
property ain86_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@86)

Address: 11272

```
property ain86_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@86)

Address: 11572

```
property ain86_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@86)

Address: 11872

```
property ain86_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN86_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@86)

Address: 12172

```
property ain86_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN86_EF_INDEX (AIN#(0:149)_EF_INDEX@86)

Address: 9172

```
property ain86_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN86_EF_READ_A (AIN#(0:149)_EF_READ_A@86)

Address: 7172

```
property ain86_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN86_EF_READ_B (AIN#(0:149)_EF_READ_B@86)

Address: 7472

```
property ain86_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN86_EF_READ_C (AIN#(0:149)_EF_READ_C@86)

Address: 7772

```
property ain86_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN86_EF_READ_D (AIN#(0:149)_EF_READ_D@86)

Address: 8072

```
property ain87_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@87)

Address: 9474

```
property ain87_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@87)

Address: 9774

```
property ain87_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@87)

Address: 10074

```
property ain87_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@87)

Address: 10374

```
property ain87_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@87)

Address: 10674

```
property ain87_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@87)

Address: 10974

```
property ain87_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@87)

Address: 11274

```
property ain87_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@87)

Address: 11574

```
property ain87_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@87)

Address: 11874

```
property ain87_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN87_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@87)

Address: 12174

```
property ain87_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN87_EF_INDEX (AIN#(0:149)_EF_INDEX@87)

Address: 9174

```
property ain87_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN87_EF_READ_A (AIN#(0:149)_EF_READ_A@87)

Address: 7174

```
property ain87_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN87_EF_READ_B (AIN#(0:149)_EF_READ_B@87)

Address: 7474

```
property ain87_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN87_EF_READ_C (AIN#(0:149)_EF_READ_C@87)

Address: 7774

```
property ain87_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN87_EF_READ_D (AIN#(0:149)_EF_READ_D@87)

Address: 8074

```
property ain88_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@88)

Address: 9476

```
property ain88_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@88)

Address: 9776

```
property ain88_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@88)

Address: 10076

```
property ain88_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@88)

Address: 10376

```
property ain88_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@88)

Address: 10676

```
property ain88_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@88)

Address: 10976

```
property ain88_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@88)

Address: 11276

```
property ain88_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@88)

Address: 11576

```
property ain88_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@88)

Address: 11876

```
property ain88_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN88_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@88)

Address: 12176

```
property ain88_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN88_EF_INDEX (AIN#(0:149)_EF_INDEX@88)

Address: 9176

```
property ain88_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN88_EF_READ_A (AIN#(0:149)_EF_READ_A@88)

Address: 7176

```
property ain88_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN88_EF_READ_B (AIN#(0:149)_EF_READ_B@88)

Address: 7476

```
property ain88_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN88_EF_READ_C (AIN#(0:149)_EF_READ_C@88)

Address: 7776

```
property ain88_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN88_EF_READ_D (AIN#(0:149)_EF_READ_D@88)

Address: 8076

```
property ain89_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@89)

Address: 9478

```
property ain89_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@89)

Address: 9778

```
property ain89_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@89)

Address: 10078

```
property ain89_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@89)

Address: 10378

```
property ain89_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@89)

Address: 10678

```
property ain89_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@89)

Address: 10978

```
property ain89_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@89)

Address: 11278

```
property ain89_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@89)

Address: 11578

```
property ain89_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@89)

Address: 11878

```
property ain89_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN89_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@89)

Address: 12178

```
property ain89_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN89_EF_INDEX (AIN#(0:149)_EF_INDEX@89)

Address: 9178

```
property ain89_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN89_EF_READ_A (AIN#(0:149)_EF_READ_A@89)

Address: 7178

```
property ain89_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN89_EF_READ_B (AIN#(0:149)_EF_READ_B@89)

Address: 7478

```
property ain89_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN89_EF_READ_C (AIN#(0:149)_EF_READ_C@89)

Address: 7778

```
property ain89_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN89_EF_READ_D (AIN#(0:149)_EF_READ_D@89)

Address: 8078

```
property ain8_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@8)

Address: 9316

```
property ain8_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@8)

Address: 9616

```
property ain8_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@8)

Address: 9916

```
property ain8_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@8)

Address: 10216

```
property ain8_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@8)

Address: 10516

```
property ain8_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@8)

Address: 10816

```
property ain8_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@8)

Address: 11116

```
property ain8_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@8)

Address: 11416

```
property ain8_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@8)

Address: 11716

```
property ain8_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN8_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@8)

Address: 12016

```
property ain8_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN8_EF_INDEX (AIN#(0:149)_EF_INDEX@8)

Address: 9016

```
property ain8_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN8_EF_READ_A (AIN#(0:149)_EF_READ_A@8)

Address: 7016

```
property ain8_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN8_EF_READ_B (AIN#(0:149)_EF_READ_B@8)

Address: 7316

```
property ain8_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN8_EF_READ_C (AIN#(0:149)_EF_READ_C@8)

Address: 7616

```
property ain8_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN8_EF_READ_D (AIN#(0:149)_EF_READ_D@8)

Address: 7916

```
property ain90_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@90)

Address: 9480

```
property ain90_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@90)

Address: 9780

```
property ain90_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@90)

Address: 10080

```
property ain90_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@90)

Address: 10380

```
property ain90_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@90)

Address: 10680

```
property ain90_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@90)

Address: 10980

```
property ain90_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@90)

Address: 11280

```
property ain90_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@90)

Address: 11580

```
property ain90_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@90)

Address: 11880

```
property ain90_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN90_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@90)

Address: 12180

```
property ain90_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN90_EF_INDEX (AIN#(0:149)_EF_INDEX@90)

Address: 9180

```
property ain90_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN90_EF_READ_A (AIN#(0:149)_EF_READ_A@90)

Address: 7180

```
property ain90_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN90_EF_READ_B (AIN#(0:149)_EF_READ_B@90)

Address: 7480

```
property ain90_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN90_EF_READ_C (AIN#(0:149)_EF_READ_C@90)

Address: 7780

```
property ain90_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN90_EF_READ_D (AIN#(0:149)_EF_READ_D@90)

Address: 8080

```
property ain91_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@91)

Address: 9482

```
property ain91_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@91)

Address: 9782

```
property ain91_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@91)

Address: 10082

```
property ain91_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@91)

Address: 10382

```
property ain91_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@91)

Address: 10682

```
property ain91_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@91)

Address: 10982

```
property ain91_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@91)

Address: 11282

```
property ain91_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@91)

Address: 11582

```
property ain91_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@91)

Address: 11882

```
property ain91_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN91_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@91)

Address: 12182

```
property ain91_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN91_EF_INDEX (AIN#(0:149)_EF_INDEX@91)

Address: 9182

```
property ain91_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN91_EF_READ_A (AIN#(0:149)_EF_READ_A@91)

Address: 7182

```
property ain91_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN91_EF_READ_B (AIN#(0:149)_EF_READ_B@91)

Address: 7482

```
property ain91_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN91_EF_READ_C (AIN#(0:149)_EF_READ_C@91)

Address: 7782

```
property ain91_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN91_EF_READ_D (AIN#(0:149)_EF_READ_D@91)

Address: 8082

```
property ain92_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@92)

Address: 9484

```
property ain92_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@92)

Address: 9784

```
property ain92_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@92)

Address: 10084

```
property ain92_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@92)

Address: 10384

```
property ain92_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@92)

Address: 10684

```
property ain92_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@92)

Address: 10984

```
property ain92_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@92)

Address: 11284

```
property ain92_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@92)

Address: 11584

```
property ain92_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@92)

Address: 11884

```
property ain92_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN92_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@92)

Address: 12184

```
property ain92_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN92_EF_INDEX (AIN#(0:149)_EF_INDEX@92)

Address: 9184

```
property ain92_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN92_EF_READ_A (AIN#(0:149)_EF_READ_A@92)

Address: 7184

```
property ain92_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN92_EF_READ_B (AIN#(0:149)_EF_READ_B@92)

Address: 7484

```
property ain92_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN92_EF_READ_C (AIN#(0:149)_EF_READ_C@92)

Address: 7784

```
property ain92_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN92_EF_READ_D (AIN#(0:149)_EF_READ_D@92)

Address: 8084

```
property ain93_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@93)

Address: 9486

```
property ain93_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@93)

Address: 9786

```
property ain93_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@93)

Address: 10086

```
property ain93_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@93)

Address: 10386

```
property ain93_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@93)

Address: 10686

```
property ain93_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@93)

Address: 10986

```
property ain93_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@93)

Address: 11286

```
property ain93_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@93)

Address: 11586

```
property ain93_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@93)

Address: 11886

```
property ain93_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN93_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@93)

Address: 12186

```
property ain93_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN93_EF_INDEX (AIN#(0:149)_EF_INDEX@93)

Address: 9186

```
property ain93_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN93_EF_READ_A (AIN#(0:149)_EF_READ_A@93)

Address: 7186

```
property ain93_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN93_EF_READ_B (AIN#(0:149)_EF_READ_B@93)

Address: 7486

```
property ain93_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN93_EF_READ_C (AIN#(0:149)_EF_READ_C@93)

Address: 7786

```
property ain93_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN93_EF_READ_D (AIN#(0:149)_EF_READ_D@93)

Address: 8086

```
property ain94_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@94)

Address: 9488

```
property ain94_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@94)

Address: 9788

```
property ain94_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@94)

Address: 10088

```
property ain94_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@94)

Address: 10388

```
property ain94_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@94)

Address: 10688

```
property ain94_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@94)

Address: 10988

```
property ain94_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@94)

Address: 11288

```
property ain94_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@94)

Address: 11588

```
property ain94_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@94)

Address: 11888

```
property ain94_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN94_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@94)

Address: 12188

```
property ain94_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN94_EF_INDEX (AIN#(0:149)_EF_INDEX@94)

Address: 9188

```
property ain94_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN94_EF_READ_A (AIN#(0:149)_EF_READ_A@94)

Address: 7188

```
property ain94_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN94_EF_READ_B (AIN#(0:149)_EF_READ_B@94)

Address: 7488

```
property ain94_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN94_EF_READ_C (AIN#(0:149)_EF_READ_C@94)

Address: 7788

```
property ain94_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN94_EF_READ_D (AIN#(0:149)_EF_READ_D@94)

Address: 8088

```
property ain95_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@95)

Address: 9490

```
property ain95_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@95)

Address: 9790

```
property ain95_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@95)

Address: 10090

```
property ain95_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@95)

Address: 10390

```
property ain95_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@95)

Address: 10690

```
property ain95_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@95)

Address: 10990

```
property ain95_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@95)

Address: 11290

```
property ain95_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@95)

Address: 11590

```
property ain95_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@95)

Address: 11890

```
property ain95_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN95_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@95)

Address: 12190

```
property ain95_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN95_EF_INDEX (AIN#(0:149)_EF_INDEX@95)

Address: 9190

```
property ain95_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN95_EF_READ_A (AIN#(0:149)_EF_READ_A@95)

Address: 7190

```
property ain95_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN95_EF_READ_B (AIN#(0:149)_EF_READ_B@95)

Address: 7490

```
property ain95_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN95_EF_READ_C (AIN#(0:149)_EF_READ_C@95)

Address: 7790

```
property ain95_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN95_EF_READ_D (AIN#(0:149)_EF_READ_D@95)

Address: 8090

```
property ain96_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@96)

Address: 9492

```
property ain96_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@96)

Address: 9792

```
property ain96_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@96)

Address: 10092

```
property ain96_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@96)

Address: 10392

```
property ain96_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@96)

Address: 10692

```
property ain96_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@96)

Address: 10992

```
property ain96_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@96)

Address: 11292

```
property ain96_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@96)

Address: 11592

```
property ain96_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@96)

Address: 11892

```
property ain96_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN96_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@96)

Address: 12192

```
property ain96_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN96_EF_INDEX (AIN#(0:149)_EF_INDEX@96)

Address: 9192

```
property ain96_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN96_EF_READ_A (AIN#(0:149)_EF_READ_A@96)

Address: 7192

```
property ain96_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN96_EF_READ_B (AIN#(0:149)_EF_READ_B@96)

Address: 7492

```
property ain96_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN96_EF_READ_C (AIN#(0:149)_EF_READ_C@96)

Address: 7792

```
property ain96_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN96_EF_READ_D (AIN#(0:149)_EF_READ_D@96)

Address: 8092

```
property ain97_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@97)

Address: 9494

```
property ain97_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@97)

Address: 9794

```
property ain97_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@97)

Address: 10094

```
property ain97_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@97)

Address: 10394

```
property ain97_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@97)

Address: 10694

```
property ain97_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@97)

Address: 10994

```
property ain97_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@97)

Address: 11294

```
property ain97_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@97)

Address: 11594

```
property ain97_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@97)

Address: 11894

```
property ain97_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN97_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@97)

Address: 12194

```
property ain97_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN97_EF_INDEX (AIN#(0:149)_EF_INDEX@97)

Address: 9194

```
property ain97_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN97_EF_READ_A (AIN#(0:149)_EF_READ_A@97)

Address: 7194

```
property ain97_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN97_EF_READ_B (AIN#(0:149)_EF_READ_B@97)

Address: 7494

```
property ain97_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN97_EF_READ_C (AIN#(0:149)_EF_READ_C@97)

Address: 7794

```
property ain97_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN97_EF_READ_D (AIN#(0:149)_EF_READ_D@97)

Address: 8094

```
property ain98_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@98)

Address: 9496

```
property ain98_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@98)

Address: 9796

```
property ain98_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@98)

Address: 10096

```
property ain98_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@98)

Address: 10396

```
property ain98_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@98)

Address: 10696

```
property ain98_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@98)

Address: 10996

```
property ain98_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@98)

Address: 11296

```
property ain98_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@98)

Address: 11596

```
property ain98_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@98)

Address: 11896

```
property ain98_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN98_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@98)

Address: 12196

```
property ain98_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN98_EF_INDEX (AIN#(0:149)_EF_INDEX@98)

Address: 9196

```
property ain98_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN98_EF_READ_A (AIN#(0:149)_EF_READ_A@98)

Address: 7196

```
property ain98_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN98_EF_READ_B (AIN#(0:149)_EF_READ_B@98)

Address: 7496

```
property ain98_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN98_EF_READ_C (AIN#(0:149)_EF_READ_C@98)

Address: 7796

```
property ain98_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN98_EF_READ_D (AIN#(0:149)_EF_READ_D@98)

Address: 8096

```
property ain99_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@99)

Address: 9498

```
property ain99_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@99)

Address: 9798

```
property ain99_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@99)

Address: 10098

```
property ain99_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@99)

Address: 10398

```
property ain99_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@99)

Address: 10698

```
property ain99_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@99)

Address: 10998

```
property ain99_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@99)

Address: 11298

```
property ain99_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@99)

Address: 11598

```
property ain99_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@99)

Address: 11898

```
property ain99_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN99_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@99)

Address: 12198

```
property ain99_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN99_EF_INDEX (AIN#(0:149)_EF_INDEX@99)

Address: 9198

```
property ain99_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN99_EF_READ_A (AIN#(0:149)_EF_READ_A@99)

Address: 7198

```
property ain99_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN99_EF_READ_B (AIN#(0:149)_EF_READ_B@99)

Address: 7498

```
property ain99_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN99_EF_READ_C (AIN#(0:149)_EF_READ_C@99)

Address: 7798

```
property ain99_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN99_EF_READ_D (AIN#(0:149)_EF_READ_D@99)

Address: 8098

```
property ain9_ef_config_a: int
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_A (AIN#(0:149)_EF_CONFIG_A@9)

Address: 9318

```
property ain9_ef_config_b: int
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_B (AIN#(0:149)_EF_CONFIG_B@9)

Address: 9618

```
property ain9_ef_config_c: int
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_C (AIN#(0:149)_EF_CONFIG_C@9)

Address: 9918

```
property ain9_ef_config_d: float
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_D (AIN#(0:149)_EF_CONFIG_D@9)

Address: 10218

```
property ain9_ef_config_e: float
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_E (AIN#(0:149)_EF_CONFIG_E@9)

Address: 10518

```
property ain9_ef_config_f: float
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_F (AIN#(0:149)_EF_CONFIG_F@9)

Address: 10818

```
property ain9_ef_config_g: float
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_G (AIN#(0:149)_EF_CONFIG_G@9)

Address: 11118

```
property ain9_ef_config_h: float
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_H (AIN#(0:149)_EF_CONFIG_H@9)

Address: 11418

```
property ain9_ef_config_i: float
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_I (AIN#(0:149)_EF_CONFIG_I@9)

Address: 11718

```
property ain9_ef_config_j: float
```

Function dependent on selected feature index.

Register: AIN9_EF_CONFIG_J (AIN#(0:149)_EF_CONFIG_J@9)

Address: 12018

```
property ain9_ef_index: int
```

Specify the desired extended feature for this analog input with the index value. List of index values: 0=None(disabled); 1=Offset and Slope; 3=Max/Min/Avg; 4=Resistance; 5=Average and Threshold; 10=RMS Flex; 11=FlexRMS; 20=Thermocouple type E; 21=Thermocouple type J; 22=Thermocouple type K; 23=Thermocouple type R; 24=Thermocouple type T; 25=Thermocouple type S; 27=Thermocouple type N; 28=Thermocouple type B; 30=Thermocouple type C; 40=RTD model PT100; 41=RTD model PT500; 42=RTD model PT1000; 50=Thermistor Steinhart-Hart; 51=Thermistor Beta.

Register: AIN9_EF_INDEX (AIN#(0:149)_EF_INDEX@9)

Address: 9018

```
property ain9_ef_read_a: float
```

Function dependent on selected feature index.

Register: AIN9_EF_READ_A (AIN#(0:149)_EF_READ_A@9)

Address: 7018

```
property ain9_ef_read_b: float
```

Function dependent on selected feature index.

Register: AIN9_EF_READ_B (AIN#(0:149)_EF_READ_B@9)

Address: 7318

```
property ain9_ef_read_c: float
```

Function dependent on selected feature index.

Register: AIN9_EF_READ_C (AIN#(0:149)_EF_READ_C@9)

Address: 7618

```
property ain9_ef_read_d: float
```

Function dependent on selected feature index.

Register: AIN9_EF_READ_D (AIN#(0:149)_EF_READ_D@9)

Address: 7918

```
property ain_all_ef_index: int
```

Write 0 to deactivate AIN_EF on all AINs. No other values may be written to this register. Reads will return the AIN_EF index if all 128 AINs are set to the same value. If values are not the same returns 0xFFFF (65535).

Register: AIN_ALL_EF_INDEX

Address: 43906

```
property ain_all_negative_ch: int
```

A write to this global parameter affects all AIN. Writing 1 will set all AINs to differential. Writing 199 will set all AINs to single-ended. A read will return 1 if all AINs are set to differential and 199 if all AINs are set to single-ended. If AIN configurations are not consistent 0xFFFF will be returned.

Register: AIN_ALL_NEGATIVE_CH

Address: 43902

```
property ain_all_range: float
```

A write to this global parameter affects all AIN. A read will return the correct setting if all channels are set the same, but otherwise will return -9999.

Register: AIN_ALL_RANGE

Address: 43900

```
property ain_all_resolution_index: int
```

The resolution index for command-response and AIN-EF readings. A larger resolution index generally results in lower noise and longer sample times. A write to this global parameter affects all AIN. A read will return the correct setting if all channels are set the same, but otherwise will return 0xFFFF.

Register: AIN_ALL_RESOLUTION_INDEX

Address: 43903

```
property ain_all_settling_us: float
```

Settling time for command-response and AIN-EF readings. A write to this global parameter affects all AIN. A read will return the correct setting if all channels are set the same, but otherwise will return -9999. Max is 50,000 us.

Register: AIN_ALL_SETTLING_US

Address: 43904

```
property ain_channel_enable: int
```

T8 Only. This register is a bitmask representing the enable states of the analog inputs.

Register: AIN_CHANNEL_ENABLE

Address: 43700

```
property ain_sampling_rate_actual_hz: float
```

T8 Only. Returns the AIN system's actual sampling rate.

Register: AIN_SAMPLING_RATE_ACTUAL_HZ

Address: 43712

```
property ain_sampling_rate_hz: float
```

T8 Only. Writing to this register sets the analog sampling rate. Not all rates are possible. Read AIN_SAMPLING_RATE_ACTUAL_HZ to get the actual rate.

Register: AIN_SAMPLING_RATE_HZ

Address: 43710

```
property asynch_baud: int
```

The symbol rate that will be used for communication. 9600 is typical. Up to 38400 works, but heavily loads the T7's processor.

Register: ASYNCH_BAUD

Address: 5420

```
property asynch_data_rx: int
```

Read received data from here.

This register is a buffer. Underrun behavior - fill with zeros.

Register: ASYNCH_DATA_RX

Address: 5495

```
property asynch_data_tx: int
```

Write data to be transmitted here. This register is a buffer.

Register: ASYNCH_DATA_TX

Address: 5490

```
property asynch_enable: int
```

1 = Turn on Asynch. Configures timing hardware, DIO lines and allocates the receiving buffer.

Register: ASYNCH_ENABLE

Address: 5400

```
property asynch_num_bytes_rx: int
```

The number of data bytes that have been received.

Register: ASYNCH_NUM_BYTES_RX

Address: 5435

```
property asynch_num_bytes_tx: int
```

The number of bytes to be transmitted after writing to GO. Max is 256.

Register: ASYNCH_NUM_BYTES_TX

Address: 5440

```
property asynch_num_data_bits: int
```

The number of data bits per frame. 0-8, 0=8.

Register: ASYNCH_NUM_DATA_BITS

Address: 5415

```
property asynch_num_parity_errors: int
```

The number of parity errors that have been detected. Cleared when UART is enabled. Can also be cleared by writing 0.

Register: ASYNCH_NUM_PARITY_ERRORS

Address: 5465

```
property asynch_num_stop_bits: int
```

The number of stop bits. Values: 0 = zero stop bits, 1 = one stop bit, 2 = two stop bits.

Register: ASYNCH_NUM_STOP_BITS

Address: 5455

```
property asynch_parity: int
```

Parity setting: 0=none, 1=odd, 2=even.

Register: ASYNCH_PARITY

Address: 5460

```
property asynch_rx_buffer_size_bytes: int
```

Number of bytes to use for the receiving buffer. Max is 2048. 0 = 200.

Register: ASYNCH_RX_BUFFER_SIZE_BYTES

Address: 5430

```
property asynch_rx_dionum: int
```

The DIO line that will receive data. (RX)

Register: ASYNCH_RX_DIONUM

Address: 5405

```
property asynch_tx_dionum: int
```

The DIO line that will transmit data. (TX)

Register: ASYNCH_TX_DIONUM

Address: 5410

```
property asynch_tx_go: int
```

Write a 1 to this register to initiate a transmission.

Register: ASYNCH_TX_GO

Address: 5450

```
property bootloader_version: float
```

The bootloader version installed on the main processor.

Register: BOOTLOADER_VERSION

Address: 60006

```
property calibration_constants_status: float
```

T8 Only. Returns the status of the calibration data saved in flash. 1 = calibration looks good. 0 = calibration invalid.

Register: CALIBRATION_CONSTANTS_STATUS

Address: 60091

```
property calibration_source: int
```

T8 Only. Sets and reads the source of the current set of calibration constants. 0 = no calibration. 1 = binary calibration, 2 = Calibration set from Flash, 3 = Nominal Calibration. When setting to 2, the constants will be checked and if invalid 1 will be used instead.

Register: CALIBRATION_SOURCE

Address: 60092

```
property cio0: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: CIO0 (CIO#(0:3)@0)

Address: 2016

```
property cio1: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: CIO1 (CIO#(0:3)@1)

Address: 2017

```
property cio2: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: CIO2 (CIO#(0:3)@2)

Address: 2018

```
property cio3: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: CIO3 (CIO#(0:3)@3)

Address: 2019

```
property cio_direction: int
```

Read or write the direction of the 4 bits of CIO in a single binary-encoded value. 0=Input and 1=Output. The upper 8-bits of this value are inhibits.

Register: CIO_DIRECTION

Address: 2602

```
property cio_mio_state: int
```

Read or write the state of the 12 bits of CIO-MIO in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. Reading lines set to output returns the current logic levels on the terminals, not necessarily the output states written.

Register: CIO_MIO_STATE

Address: 2582

```
property cio_state: int
```

Read or write the state of the 4 bits of CIO in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. Reading lines set to output returns the current logic levels on the terminals, not necessarily the output states written. The upper 8-bits of this value are inhibits.

Register: CIO_STATE

Address: 2502

```
property cleanse: int
```

Writing 0x5317052E to this register will trigger a system cleanse.

Register: CLEANSE

Address: 49090

```
property core_timer: int
```

Internal 32-bit system timer running at 1/2 core speed, thus normally 80M/2 => 40 MHz.

Register: CORE_TIMER

Address: 61520

```
property current_source_10ua_cal_value: float
```

Fixed current source value in Amps for the 10UA terminal. This value is stored during factory calibration, it is not a current reading. Using the equation V=IR, with a known current and voltage, it is possible to calculate resistance of RTDs.

Register: CURRENT_SOURCE_10UA_CAL_VALUE

Address: 1900

```
property current_source_200ua_cal_value: float
```

Fixed current source value in Amps for the 200UA terminal. This value is stored during factory calibration, it is not a current reading. Using the equation V=IR, with a known current and voltage, it is possible to calculate resistance of RTDs.

Register: CURRENT_SOURCE_200UA_CAL_VALUE

Address: 1902

```
property dac0: float
```

Pass a voltage for the specified analog output.

Register: DAC0 (DAC#(0:1)@0)

Address: 1000

```
property dac0_binary: int
```

Writes binary values to the DACs. Binary values are 16-bit. If the DAC's resolution is less than 16 then the lower bits are ignored. 0 = lowest output, 65535 = highest output.

Register: DAC0_BINARY (DAC#(0:1)_BINARY@0)

Address: 51000

```
property dac1: float
```

Pass a voltage for the specified analog output.

Register: DAC1 (DAC#(0:1)@1)

Address: 1002

```
property dac1_binary: int
```

Writes binary values to the DACs. Binary values are 16-bit. If the DAC's resolution is less than 16 then the lower bits are ignored. 0 = lowest output, 65535 = highest output.

Register: DAC1_BINARY (DAC#(0:1)_BINARY@1)

Address: 51002

```
property dac1_frequency_out_enable: int
```

0 = off, 1 = output 10 Hz signal on DAC1. The signal will be a square wave with peaks of 0 and 3.3V. Note that writing to DAC1 or enabling a stream out which is targeting DAC1 will disable this feature.

Register: DAC1_FREQUENCY_OUT_ENABLE

Address: 61532

```
property device_name_default: str
```

Reads return the current device name. Writes update the default and current device name. A reboot is necessary to update the name reported by NBNS. Up to 49 characters, cannot contain periods.

Register: DEVICE_NAME_DEFAULT

Address: 60500

```
property device_reset_dbg_reg: int
```

Bitmask register that indicates why a device was last re-set. bit0: Power-on reset, bit1: Brown-out, bit2: Wake from idle, bit3: Wake from sleep, bit4: watchdog timer time-out, bit5: reserved, bit6: Software reset, bit7: external reset (MCLR), bit8: Voltage regulator standby, bit9: Configuration mismatch.

Register: DEVICE_RESET_DBG_REG

Address: 60014

```
property dio0_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO0_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@0)

Address: 44300

```
property dio0_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO0_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@0)

Address: 44400

```
property dio0_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO0_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@0)

Address: 44500

```
property dio0_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO0_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@0)

Address: 44600

```
property dio0_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO0_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@0)

Address: 45000

```
property dio0_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO0_EF_ENABLE (DIO#(0:22)_EF_ENABLE@0)

Address: 44000

```
property dio0_ef_index: int
```

An index to specify the feature you want.

Register: DIO0_EF_INDEX (DIO#(0:22)_EF_INDEX@0)

Address: 44100

```
property dio0_ef_options: int
```

Function dependent on selected feature index.

Register: DIO0_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@0)

Address: 44200

```
property dio0_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO0_EF_READ_A (DIO#(0:22)_EF_READ_A@0)

Address: 3000

```
property dio0_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO0_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@0)

Address: 3100

Type:
Reads the same value as DIO#(0

```
property dio0_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO0_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@0)

Address: 3500

```
property dio0_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO0_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@0)

Address: 3600

```
property dio0_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO0_EF_READ_B (DIO#(0:22)_EF_READ_B@0)

Address: 3200

```
property dio0_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO0_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@0)

Address: 3700

```
property dio10_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO10_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@10)

Address: 44320

```
property dio10_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO10_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@10)

Address: 44420

```
property dio10_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO10_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@10)

Address: 44520

```
property dio10_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO10_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@10)

Address: 44620

```
property dio10_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO10_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@10)

Address: 45020

```
property dio10_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO10_EF_ENABLE (DIO#(0:22)_EF_ENABLE@10)

Address: 44020

```
property dio10_ef_index: int
```

An index to specify the feature you want.

Register: DIO10_EF_INDEX (DIO#(0:22)_EF_INDEX@10)

Address: 44120

```
property dio10_ef_options: int
```

Function dependent on selected feature index.

Register: DIO10_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@10)

Address: 44220

```
property dio10_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO10_EF_READ_A (DIO#(0:22)_EF_READ_A@10)

Address: 3020

```
property dio10_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO10_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@10)

Address: 3120

Type:
Reads the same value as DIO#(0

```
property dio10_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO10_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@10)

Address: 3520

```
property dio10_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO10_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@10)

Address: 3620

```
property dio10_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO10_EF_READ_B (DIO#(0:22)_EF_READ_B@10)

Address: 3220

```
property dio10_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO10_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@10)

Address: 3720

```
property dio11_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO11_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@11)

Address: 44322

```
property dio11_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO11_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@11)

Address: 44422

```
property dio11_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO11_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@11)

Address: 44522

```
property dio11_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO11_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@11)

Address: 44622

```
property dio11_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO11_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@11)

Address: 45022

```
property dio11_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO11_EF_ENABLE (DIO#(0:22)_EF_ENABLE@11)

Address: 44022

```
property dio11_ef_index: int
```

An index to specify the feature you want.

Register: DIO11_EF_INDEX (DIO#(0:22)_EF_INDEX@11)

Address: 44122

```
property dio11_ef_options: int
```

Function dependent on selected feature index.

Register: DIO11_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@11)

Address: 44222

```
property dio11_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO11_EF_READ_A (DIO#(0:22)_EF_READ_A@11)

Address: 3022

```
property dio11_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO11_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@11)

Address: 3122

Type:
Reads the same value as DIO#(0

```
property dio11_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO11_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@11)

Address: 3522

```
property dio11_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO11_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@11)

Address: 3622

```
property dio11_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO11_EF_READ_B (DIO#(0:22)_EF_READ_B@11)

Address: 3222

```
property dio11_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO11_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@11)

Address: 3722

```
property dio12_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO12_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@12)

Address: 44324

```
property dio12_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO12_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@12)

Address: 44424

```
property dio12_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO12_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@12)

Address: 44524

```
property dio12_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO12_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@12)

Address: 44624

```
property dio12_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO12_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@12)

Address: 45024

```
property dio12_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO12_EF_ENABLE (DIO#(0:22)_EF_ENABLE@12)

Address: 44024

```
property dio12_ef_index: int
```

An index to specify the feature you want.

Register: DIO12_EF_INDEX (DIO#(0:22)_EF_INDEX@12)

Address: 44124

```
property dio12_ef_options: int
```

Function dependent on selected feature index.

Register: DIO12_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@12)

Address: 44224

```
property dio12_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO12_EF_READ_A (DIO#(0:22)_EF_READ_A@12)

Address: 3024

```
property dio12_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO12_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@12)

Address: 3124

Type:
Reads the same value as DIO#(0

```
property dio12_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO12_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@12)

Address: 3524

```
property dio12_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO12_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@12)

Address: 3624

```
property dio12_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO12_EF_READ_B (DIO#(0:22)_EF_READ_B@12)

Address: 3224

```
property dio12_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO12_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@12)

Address: 3724

```
property dio13_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO13_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@13)

Address: 44326

```
property dio13_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO13_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@13)

Address: 44426

```
property dio13_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO13_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@13)

Address: 44526

```
property dio13_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO13_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@13)

Address: 44626

```
property dio13_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO13_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@13)

Address: 45026

```
property dio13_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO13_EF_ENABLE (DIO#(0:22)_EF_ENABLE@13)

Address: 44026

```
property dio13_ef_index: int
```

An index to specify the feature you want.

Register: DIO13_EF_INDEX (DIO#(0:22)_EF_INDEX@13)

Address: 44126

```
property dio13_ef_options: int
```

Function dependent on selected feature index.

Register: DIO13_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@13)

Address: 44226

```
property dio13_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO13_EF_READ_A (DIO#(0:22)_EF_READ_A@13)

Address: 3026

```
property dio13_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO13_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@13)

Address: 3126

Type:
Reads the same value as DIO#(0

```
property dio13_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO13_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@13)

Address: 3526

```
property dio13_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO13_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@13)

Address: 3626

```
property dio13_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO13_EF_READ_B (DIO#(0:22)_EF_READ_B@13)

Address: 3226

```
property dio13_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO13_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@13)

Address: 3726

```
property dio14_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO14_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@14)

Address: 44328

```
property dio14_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO14_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@14)

Address: 44428

```
property dio14_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO14_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@14)

Address: 44528

```
property dio14_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO14_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@14)

Address: 44628

```
property dio14_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO14_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@14)

Address: 45028

```
property dio14_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO14_EF_ENABLE (DIO#(0:22)_EF_ENABLE@14)

Address: 44028

```
property dio14_ef_index: int
```

An index to specify the feature you want.

Register: DIO14_EF_INDEX (DIO#(0:22)_EF_INDEX@14)

Address: 44128

```
property dio14_ef_options: int
```

Function dependent on selected feature index.

Register: DIO14_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@14)

Address: 44228

```
property dio14_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO14_EF_READ_A (DIO#(0:22)_EF_READ_A@14)

Address: 3028

```
property dio14_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO14_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@14)

Address: 3128

Type:
Reads the same value as DIO#(0

```
property dio14_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO14_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@14)

Address: 3528

```
property dio14_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO14_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@14)

Address: 3628

```
property dio14_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO14_EF_READ_B (DIO#(0:22)_EF_READ_B@14)

Address: 3228

```
property dio14_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO14_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@14)

Address: 3728

```
property dio15_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO15_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@15)

Address: 44330

```
property dio15_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO15_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@15)

Address: 44430

```
property dio15_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO15_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@15)

Address: 44530

```
property dio15_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO15_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@15)

Address: 44630

```
property dio15_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO15_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@15)

Address: 45030

```
property dio15_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO15_EF_ENABLE (DIO#(0:22)_EF_ENABLE@15)

Address: 44030

```
property dio15_ef_index: int
```

An index to specify the feature you want.

Register: DIO15_EF_INDEX (DIO#(0:22)_EF_INDEX@15)

Address: 44130

```
property dio15_ef_options: int
```

Function dependent on selected feature index.

Register: DIO15_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@15)

Address: 44230

```
property dio15_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO15_EF_READ_A (DIO#(0:22)_EF_READ_A@15)

Address: 3030

```
property dio15_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO15_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@15)

Address: 3130

Type:
Reads the same value as DIO#(0

```
property dio15_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO15_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@15)

Address: 3530

```
property dio15_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO15_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@15)

Address: 3630

```
property dio15_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO15_EF_READ_B (DIO#(0:22)_EF_READ_B@15)

Address: 3230

```
property dio15_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO15_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@15)

Address: 3730

```
property dio16_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO16_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@16)

Address: 44332

```
property dio16_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO16_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@16)

Address: 44432

```
property dio16_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO16_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@16)

Address: 44532

```
property dio16_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO16_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@16)

Address: 44632

```
property dio16_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO16_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@16)

Address: 45032

```
property dio16_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO16_EF_ENABLE (DIO#(0:22)_EF_ENABLE@16)

Address: 44032

```
property dio16_ef_index: int
```

An index to specify the feature you want.

Register: DIO16_EF_INDEX (DIO#(0:22)_EF_INDEX@16)

Address: 44132

```
property dio16_ef_options: int
```

Function dependent on selected feature index.

Register: DIO16_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@16)

Address: 44232

```
property dio16_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO16_EF_READ_A (DIO#(0:22)_EF_READ_A@16)

Address: 3032

```
property dio16_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO16_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@16)

Address: 3132

Type:
Reads the same value as DIO#(0

```
property dio16_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO16_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@16)

Address: 3532

```
property dio16_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO16_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@16)

Address: 3632

```
property dio16_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO16_EF_READ_B (DIO#(0:22)_EF_READ_B@16)

Address: 3232

```
property dio16_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO16_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@16)

Address: 3732

```
property dio17_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO17_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@17)

Address: 44334

```
property dio17_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO17_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@17)

Address: 44434

```
property dio17_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO17_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@17)

Address: 44534

```
property dio17_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO17_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@17)

Address: 44634

```
property dio17_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO17_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@17)

Address: 45034

```
property dio17_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO17_EF_ENABLE (DIO#(0:22)_EF_ENABLE@17)

Address: 44034

```
property dio17_ef_index: int
```

An index to specify the feature you want.

Register: DIO17_EF_INDEX (DIO#(0:22)_EF_INDEX@17)

Address: 44134

```
property dio17_ef_options: int
```

Function dependent on selected feature index.

Register: DIO17_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@17)

Address: 44234

```
property dio17_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO17_EF_READ_A (DIO#(0:22)_EF_READ_A@17)

Address: 3034

```
property dio17_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO17_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@17)

Address: 3134

Type:
Reads the same value as DIO#(0

```
property dio17_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO17_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@17)

Address: 3534

```
property dio17_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO17_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@17)

Address: 3634

```
property dio17_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO17_EF_READ_B (DIO#(0:22)_EF_READ_B@17)

Address: 3234

```
property dio17_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO17_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@17)

Address: 3734

```
property dio18_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO18_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@18)

Address: 44336

```
property dio18_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO18_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@18)

Address: 44436

```
property dio18_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO18_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@18)

Address: 44536

```
property dio18_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO18_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@18)

Address: 44636

```
property dio18_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO18_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@18)

Address: 45036

```
property dio18_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO18_EF_ENABLE (DIO#(0:22)_EF_ENABLE@18)

Address: 44036

```
property dio18_ef_index: int
```

An index to specify the feature you want.

Register: DIO18_EF_INDEX (DIO#(0:22)_EF_INDEX@18)

Address: 44136

```
property dio18_ef_options: int
```

Function dependent on selected feature index.

Register: DIO18_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@18)

Address: 44236

```
property dio18_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO18_EF_READ_A (DIO#(0:22)_EF_READ_A@18)

Address: 3036

```
property dio18_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO18_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@18)

Address: 3136

Type:
Reads the same value as DIO#(0

```
property dio18_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO18_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@18)

Address: 3536

```
property dio18_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO18_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@18)

Address: 3636

```
property dio18_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO18_EF_READ_B (DIO#(0:22)_EF_READ_B@18)

Address: 3236

```
property dio18_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO18_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@18)

Address: 3736

```
property dio19_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO19_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@19)

Address: 44338

```
property dio19_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO19_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@19)

Address: 44438

```
property dio19_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO19_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@19)

Address: 44538

```
property dio19_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO19_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@19)

Address: 44638

```
property dio19_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO19_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@19)

Address: 45038

```
property dio19_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO19_EF_ENABLE (DIO#(0:22)_EF_ENABLE@19)

Address: 44038

```
property dio19_ef_index: int
```

An index to specify the feature you want.

Register: DIO19_EF_INDEX (DIO#(0:22)_EF_INDEX@19)

Address: 44138

```
property dio19_ef_options: int
```

Function dependent on selected feature index.

Register: DIO19_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@19)

Address: 44238

```
property dio19_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO19_EF_READ_A (DIO#(0:22)_EF_READ_A@19)

Address: 3038

```
property dio19_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO19_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@19)

Address: 3138

Type:
Reads the same value as DIO#(0

```
property dio19_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO19_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@19)

Address: 3538

```
property dio19_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO19_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@19)

Address: 3638

```
property dio19_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO19_EF_READ_B (DIO#(0:22)_EF_READ_B@19)

Address: 3238

```
property dio19_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO19_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@19)

Address: 3738

```
property dio1_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO1_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@1)

Address: 44302

```
property dio1_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO1_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@1)

Address: 44402

```
property dio1_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO1_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@1)

Address: 44502

```
property dio1_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO1_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@1)

Address: 44602

```
property dio1_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO1_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@1)

Address: 45002

```
property dio1_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO1_EF_ENABLE (DIO#(0:22)_EF_ENABLE@1)

Address: 44002

```
property dio1_ef_index: int
```

An index to specify the feature you want.

Register: DIO1_EF_INDEX (DIO#(0:22)_EF_INDEX@1)

Address: 44102

```
property dio1_ef_options: int
```

Function dependent on selected feature index.

Register: DIO1_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@1)

Address: 44202

```
property dio1_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO1_EF_READ_A (DIO#(0:22)_EF_READ_A@1)

Address: 3002

```
property dio1_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO1_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@1)

Address: 3102

Type:
Reads the same value as DIO#(0

```
property dio1_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO1_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@1)

Address: 3502

```
property dio1_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO1_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@1)

Address: 3602

```
property dio1_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO1_EF_READ_B (DIO#(0:22)_EF_READ_B@1)

Address: 3202

```
property dio1_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO1_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@1)

Address: 3702

```
property dio20_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO20_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@20)

Address: 44340

```
property dio20_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO20_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@20)

Address: 44440

```
property dio20_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO20_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@20)

Address: 44540

```
property dio20_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO20_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@20)

Address: 44640

```
property dio20_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO20_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@20)

Address: 45040

```
property dio20_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO20_EF_ENABLE (DIO#(0:22)_EF_ENABLE@20)

Address: 44040

```
property dio20_ef_index: int
```

An index to specify the feature you want.

Register: DIO20_EF_INDEX (DIO#(0:22)_EF_INDEX@20)

Address: 44140

```
property dio20_ef_options: int
```

Function dependent on selected feature index.

Register: DIO20_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@20)

Address: 44240

```
property dio20_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO20_EF_READ_A (DIO#(0:22)_EF_READ_A@20)

Address: 3040

```
property dio20_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO20_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@20)

Address: 3140

Type:
Reads the same value as DIO#(0

```
property dio20_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO20_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@20)

Address: 3540

```
property dio20_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO20_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@20)

Address: 3640

```
property dio20_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO20_EF_READ_B (DIO#(0:22)_EF_READ_B@20)

Address: 3240

```
property dio20_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO20_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@20)

Address: 3740

```
property dio21_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO21_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@21)

Address: 44342

```
property dio21_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO21_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@21)

Address: 44442

```
property dio21_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO21_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@21)

Address: 44542

```
property dio21_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO21_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@21)

Address: 44642

```
property dio21_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO21_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@21)

Address: 45042

```
property dio21_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO21_EF_ENABLE (DIO#(0:22)_EF_ENABLE@21)

Address: 44042

```
property dio21_ef_index: int
```

An index to specify the feature you want.

Register: DIO21_EF_INDEX (DIO#(0:22)_EF_INDEX@21)

Address: 44142

```
property dio21_ef_options: int
```

Function dependent on selected feature index.

Register: DIO21_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@21)

Address: 44242

```
property dio21_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO21_EF_READ_A (DIO#(0:22)_EF_READ_A@21)

Address: 3042

```
property dio21_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO21_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@21)

Address: 3142

Type:
Reads the same value as DIO#(0

```
property dio21_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO21_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@21)

Address: 3542

```
property dio21_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO21_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@21)

Address: 3642

```
property dio21_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO21_EF_READ_B (DIO#(0:22)_EF_READ_B@21)

Address: 3242

```
property dio21_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO21_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@21)

Address: 3742

```
property dio22_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO22_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@22)

Address: 44344

```
property dio22_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO22_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@22)

Address: 44444

```
property dio22_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO22_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@22)

Address: 44544

```
property dio22_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO22_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@22)

Address: 44644

```
property dio22_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO22_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@22)

Address: 45044

```
property dio22_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO22_EF_ENABLE (DIO#(0:22)_EF_ENABLE@22)

Address: 44044

```
property dio22_ef_index: int
```

An index to specify the feature you want.

Register: DIO22_EF_INDEX (DIO#(0:22)_EF_INDEX@22)

Address: 44144

```
property dio22_ef_options: int
```

Function dependent on selected feature index.

Register: DIO22_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@22)

Address: 44244

```
property dio22_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO22_EF_READ_A (DIO#(0:22)_EF_READ_A@22)

Address: 3044

```
property dio22_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO22_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@22)

Address: 3144

Type:
Reads the same value as DIO#(0

```
property dio22_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO22_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@22)

Address: 3544

```
property dio22_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO22_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@22)

Address: 3644

```
property dio22_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO22_EF_READ_B (DIO#(0:22)_EF_READ_B@22)

Address: 3244

```
property dio22_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO22_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@22)

Address: 3744

```
property dio2_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO2_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@2)

Address: 44304

```
property dio2_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO2_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@2)

Address: 44404

```
property dio2_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO2_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@2)

Address: 44504

```
property dio2_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO2_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@2)

Address: 44604

```
property dio2_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO2_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@2)

Address: 45004

```
property dio2_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO2_EF_ENABLE (DIO#(0:22)_EF_ENABLE@2)

Address: 44004

```
property dio2_ef_index: int
```

An index to specify the feature you want.

Register: DIO2_EF_INDEX (DIO#(0:22)_EF_INDEX@2)

Address: 44104

```
property dio2_ef_options: int
```

Function dependent on selected feature index.

Register: DIO2_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@2)

Address: 44204

```
property dio2_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO2_EF_READ_A (DIO#(0:22)_EF_READ_A@2)

Address: 3004

```
property dio2_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO2_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@2)

Address: 3104

Type:
Reads the same value as DIO#(0

```
property dio2_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO2_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@2)

Address: 3504

```
property dio2_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO2_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@2)

Address: 3604

```
property dio2_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO2_EF_READ_B (DIO#(0:22)_EF_READ_B@2)

Address: 3204

```
property dio2_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO2_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@2)

Address: 3704

```
property dio3_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO3_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@3)

Address: 44306

```
property dio3_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO3_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@3)

Address: 44406

```
property dio3_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO3_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@3)

Address: 44506

```
property dio3_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO3_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@3)

Address: 44606

```
property dio3_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO3_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@3)

Address: 45006

```
property dio3_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO3_EF_ENABLE (DIO#(0:22)_EF_ENABLE@3)

Address: 44006

```
property dio3_ef_index: int
```

An index to specify the feature you want.

Register: DIO3_EF_INDEX (DIO#(0:22)_EF_INDEX@3)

Address: 44106

```
property dio3_ef_options: int
```

Function dependent on selected feature index.

Register: DIO3_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@3)

Address: 44206

```
property dio3_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO3_EF_READ_A (DIO#(0:22)_EF_READ_A@3)

Address: 3006

```
property dio3_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO3_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@3)

Address: 3106

Type:
Reads the same value as DIO#(0

```
property dio3_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO3_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@3)

Address: 3506

```
property dio3_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO3_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@3)

Address: 3606

```
property dio3_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO3_EF_READ_B (DIO#(0:22)_EF_READ_B@3)

Address: 3206

```
property dio3_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO3_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@3)

Address: 3706

```
property dio4_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO4_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@4)

Address: 44308

```
property dio4_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO4_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@4)

Address: 44408

```
property dio4_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO4_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@4)

Address: 44508

```
property dio4_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO4_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@4)

Address: 44608

```
property dio4_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO4_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@4)

Address: 45008

```
property dio4_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO4_EF_ENABLE (DIO#(0:22)_EF_ENABLE@4)

Address: 44008

```
property dio4_ef_index: int
```

An index to specify the feature you want.

Register: DIO4_EF_INDEX (DIO#(0:22)_EF_INDEX@4)

Address: 44108

```
property dio4_ef_options: int
```

Function dependent on selected feature index.

Register: DIO4_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@4)

Address: 44208

```
property dio4_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO4_EF_READ_A (DIO#(0:22)_EF_READ_A@4)

Address: 3008

```
property dio4_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO4_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@4)

Address: 3108

Type:
Reads the same value as DIO#(0

```
property dio4_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO4_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@4)

Address: 3508

```
property dio4_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO4_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@4)

Address: 3608

```
property dio4_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO4_EF_READ_B (DIO#(0:22)_EF_READ_B@4)

Address: 3208

```
property dio4_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO4_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@4)

Address: 3708

```
property dio5_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO5_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@5)

Address: 44310

```
property dio5_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO5_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@5)

Address: 44410

```
property dio5_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO5_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@5)

Address: 44510

```
property dio5_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO5_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@5)

Address: 44610

```
property dio5_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO5_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@5)

Address: 45010

```
property dio5_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO5_EF_ENABLE (DIO#(0:22)_EF_ENABLE@5)

Address: 44010

```
property dio5_ef_index: int
```

An index to specify the feature you want.

Register: DIO5_EF_INDEX (DIO#(0:22)_EF_INDEX@5)

Address: 44110

```
property dio5_ef_options: int
```

Function dependent on selected feature index.

Register: DIO5_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@5)

Address: 44210

```
property dio5_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO5_EF_READ_A (DIO#(0:22)_EF_READ_A@5)

Address: 3010

```
property dio5_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO5_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@5)

Address: 3110

Type:
Reads the same value as DIO#(0

```
property dio5_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO5_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@5)

Address: 3510

```
property dio5_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO5_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@5)

Address: 3610

```
property dio5_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO5_EF_READ_B (DIO#(0:22)_EF_READ_B@5)

Address: 3210

```
property dio5_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO5_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@5)

Address: 3710

```
property dio6_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO6_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@6)

Address: 44312

```
property dio6_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO6_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@6)

Address: 44412

```
property dio6_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO6_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@6)

Address: 44512

```
property dio6_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO6_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@6)

Address: 44612

```
property dio6_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO6_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@6)

Address: 45012

```
property dio6_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO6_EF_ENABLE (DIO#(0:22)_EF_ENABLE@6)

Address: 44012

```
property dio6_ef_index: int
```

An index to specify the feature you want.

Register: DIO6_EF_INDEX (DIO#(0:22)_EF_INDEX@6)

Address: 44112

```
property dio6_ef_options: int
```

Function dependent on selected feature index.

Register: DIO6_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@6)

Address: 44212

```
property dio6_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO6_EF_READ_A (DIO#(0:22)_EF_READ_A@6)

Address: 3012

```
property dio6_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO6_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@6)

Address: 3112

Type:
Reads the same value as DIO#(0

```
property dio6_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO6_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@6)

Address: 3512

```
property dio6_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO6_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@6)

Address: 3612

```
property dio6_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO6_EF_READ_B (DIO#(0:22)_EF_READ_B@6)

Address: 3212

```
property dio6_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO6_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@6)

Address: 3712

```
property dio7_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO7_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@7)

Address: 44314

```
property dio7_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO7_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@7)

Address: 44414

```
property dio7_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO7_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@7)

Address: 44514

```
property dio7_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO7_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@7)

Address: 44614

```
property dio7_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO7_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@7)

Address: 45014

```
property dio7_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO7_EF_ENABLE (DIO#(0:22)_EF_ENABLE@7)

Address: 44014

```
property dio7_ef_index: int
```

An index to specify the feature you want.

Register: DIO7_EF_INDEX (DIO#(0:22)_EF_INDEX@7)

Address: 44114

```
property dio7_ef_options: int
```

Function dependent on selected feature index.

Register: DIO7_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@7)

Address: 44214

```
property dio7_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO7_EF_READ_A (DIO#(0:22)_EF_READ_A@7)

Address: 3014

```
property dio7_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO7_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@7)

Address: 3114

Type:
Reads the same value as DIO#(0

```
property dio7_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO7_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@7)

Address: 3514

```
property dio7_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO7_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@7)

Address: 3614

```
property dio7_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO7_EF_READ_B (DIO#(0:22)_EF_READ_B@7)

Address: 3214

```
property dio7_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO7_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@7)

Address: 3714

```
property dio8_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO8_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@8)

Address: 44316

```
property dio8_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO8_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@8)

Address: 44416

```
property dio8_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO8_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@8)

Address: 44516

```
property dio8_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO8_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@8)

Address: 44616

```
property dio8_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO8_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@8)

Address: 45016

```
property dio8_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO8_EF_ENABLE (DIO#(0:22)_EF_ENABLE@8)

Address: 44016

```
property dio8_ef_index: int
```

An index to specify the feature you want.

Register: DIO8_EF_INDEX (DIO#(0:22)_EF_INDEX@8)

Address: 44116

```
property dio8_ef_options: int
```

Function dependent on selected feature index.

Register: DIO8_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@8)

Address: 44216

```
property dio8_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO8_EF_READ_A (DIO#(0:22)_EF_READ_A@8)

Address: 3016

```
property dio8_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO8_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@8)

Address: 3116

Type:
Reads the same value as DIO#(0

```
property dio8_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO8_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@8)

Address: 3516

```
property dio8_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO8_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@8)

Address: 3616

```
property dio8_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO8_EF_READ_B (DIO#(0:22)_EF_READ_B@8)

Address: 3216

```
property dio8_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO8_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@8)

Address: 3716

```
property dio9_ef_config_a: int
```

Function dependent on selected feature index.

Register: DIO9_EF_CONFIG_A (DIO#(0:22)_EF_CONFIG_A@9)

Address: 44318

```
property dio9_ef_config_b: int
```

Function dependent on selected feature index.

Register: DIO9_EF_CONFIG_B (DIO#(0:22)_EF_CONFIG_B@9)

Address: 44418

```
property dio9_ef_config_c: int
```

Function dependent on selected feature index.

Register: DIO9_EF_CONFIG_C (DIO#(0:22)_EF_CONFIG_C@9)

Address: 44518

```
property dio9_ef_config_d: int
```

Function dependent on selected feature index.

Register: DIO9_EF_CONFIG_D (DIO#(0:22)_EF_CONFIG_D@9)

Address: 44618

```
property dio9_ef_easy_frequency_in: float
```

Deprecated. Automatically configures the associated DIO_EF for feature index 11. If already configured for index 11, reads the result and resets the DIO_EF.

Register: DIO9_EF_EASY_FREQUENCY_IN (DIO#(0:22)_EF_EASY_FREQUENCY_IN@9)

Address: 45018

```
property dio9_ef_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration. Note that DIO-EF reads work when disabled and do not return an error.

Register: DIO9_EF_ENABLE (DIO#(0:22)_EF_ENABLE@9)

Address: 44018

```
property dio9_ef_index: int
```

An index to specify the feature you want.

Register: DIO9_EF_INDEX (DIO#(0:22)_EF_INDEX@9)

Address: 44118

```
property dio9_ef_options: int
```

Function dependent on selected feature index.

Register: DIO9_EF_OPTIONS (DIO#(0:22)_EF_OPTIONS@9)

Address: 44218

```
property dio9_ef_read_a: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO9_EF_READ_A (DIO#(0:22)_EF_READ_A@9)

Address: 3018

```
property dio9_ef_read_a_and_reset: int
```

22)_EF_READ_A and forces a reset.

Register: DIO9_EF_READ_A_AND_RESET (DIO#(0:22)_EF_READ_A_AND_RESET@9)

Address: 3118

Type:
Reads the same value as DIO#(0

```
property dio9_ef_read_a_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO9_EF_READ_A_F (DIO#(0:22)_EF_READ_A_F@9)

Address: 3518

```
property dio9_ef_read_a_f_and_reset: float
```

Reads a floating point value and forces a reset. The meaning of value is dependent on selected feature index.

Register: DIO9_EF_READ_A_F_AND_RESET (DIO#(0:22)_EF_READ_A_F_AND_RESET@9)

Address: 3618

```
property dio9_ef_read_b: int
```

Reads an unsigned integer value. The meaning of the integer is dependent on selected feature index.

Register: DIO9_EF_READ_B (DIO#(0:22)_EF_READ_B@9)

Address: 3218

```
property dio9_ef_read_b_f: float
```

Reads a floating point value. The meaning of value is dependent on selected feature index.

Register: DIO9_EF_READ_B_F (DIO#(0:22)_EF_READ_B_F@9)

Address: 3718

```
property dio_direction: int
```

Read or write the direction of all digital I/O in a single binary-encoded value. 0=Input and 1=Output. Writes are filtered by the value in DIO_INHIBIT.

Register: DIO_DIRECTION

Address: 2850

```
property dio_ef_clock0_count: int
```

Current tick count of this clock. Will read between 0 and ROLL_VALUE-1.

Register: DIO_EF_CLOCK0_COUNT

Address: 44908

```
property dio_ef_clock0_divisor: int
```

Divides the core clock. Valid options: 1,2,4,8,16,32,64,256.

Register: DIO_EF_CLOCK0_DIVISOR

Address: 44901

```
property dio_ef_clock0_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration.

Register: DIO_EF_CLOCK0_ENABLE

Address: 44900

```
property dio_ef_clock0_options: int
```

Bitmask: bit0: 1 = use external clock. All other bits reserved.

Register: DIO_EF_CLOCK0_OPTIONS

Address: 44902

```
property dio_ef_clock0_roll_value: int
```

The clock will count to this value and then start over at zero. The clock pulses counted are those after the divisor. 0 results in the max roll value possible. This is a 32-bit value (0-4294967295) if using a 32-bit clock, and a 16-bit value (0-65535) if using a 16-bit clock.

Register: DIO_EF_CLOCK0_ROLL_VALUE

Address: 44904

```
property dio_ef_clock1_count: int
```

Current tick count of this clock. Will read between 0 and ROLL_VALUE-1.

Register: DIO_EF_CLOCK1_COUNT

Address: 44918

```
property dio_ef_clock1_divisor: int
```

Divides the core clock. Valid options: 1,2,4,8,16,32,64,256.

Register: DIO_EF_CLOCK1_DIVISOR

Address: 44911

```
property dio_ef_clock1_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration.

Register: DIO_EF_CLOCK1_ENABLE

Address: 44910

```
property dio_ef_clock1_options: int
```

Bitmask: bit0: 1 = use external clock. All other bits reserved.

Register: DIO_EF_CLOCK1_OPTIONS

Address: 44912

```
property dio_ef_clock1_roll_value: int
```

The clock will count to this value and then start over at zero. The clock pulses counted are those after the divisor. 0 results in the max roll value possible. This is a 32-bit value (0-4294967295) if using a 32-bit clock, and a 16-bit value (0-65535) if using a 16-bit clock.

Register: DIO_EF_CLOCK1_ROLL_VALUE

Address: 44914

```
property dio_ef_clock2_count: int
```

Current tick count of this clock. Will read between 0 and ROLL_VALUE-1.

Register: DIO_EF_CLOCK2_COUNT

Address: 44928

```
property dio_ef_clock2_divisor: int
```

Divides the core clock. Valid options: 1,2,4,8,16,32,64,256.

Register: DIO_EF_CLOCK2_DIVISOR

Address: 44921

```
property dio_ef_clock2_enable: int
```

1 = enabled. 0 = disabled. Must be disabled during configuration.

Register: DIO_EF_CLOCK2_ENABLE

Address: 44920

```
property dio_ef_clock2_options: int
```

Bitmask: bit0: 1 = use external clock. All other bits reserved.

Register: DIO_EF_CLOCK2_OPTIONS

Address: 44922

```
property dio_ef_clock2_roll_value: int
```

The clock will count to this value and then start over at zero. The clock pulses counted are those after the divisor. 0 results in the max roll value possible. This is a 32-bit value (0-4294967295) if using a 32-bit clock, and a 16-bit value (0-65535) if using a 16-bit clock.

Register: DIO_EF_CLOCK2_ROLL_VALUE

Address: 44924

```
property dio_inhibit: int
```

A single binary-encoded value where each bit determines whether _STATE, _DIRECTION or _ANALOG_ENABLE writes affect that bit of digital I/O. 0=Default=Affected, 1=Ignored.

Register: DIO_INHIBIT

Address: 2900

```
property dio_pulldown_enable: int
```

T8 Only. This register will enable pulldowns on DIO lines. This is a binary coded value where bit 0 represent FIO0, and bit 11 represents EIO3, etc. 1 = pulldown enabled, 0 = pulldown disabled. This register only affects flex-lines which can be configured as analog or digital. This register is not affected by the inhibit register.

Register: DIO_PULLDOWN_ENABLE

Address: 2870

```
property dio_pullup_disable: int
```

This register will prevent pullups from being enabled on lines set to digital input. This is a binary coded value where bit 0 represent FIO0 and bit 11 represents EIO3. 1 = pullup disabled, 0 = pullup enabled. This register is not affected by the inhibit register.

Register: DIO_PULLUP_DISABLE

Address: 2890

```
property dio_state: int
```

Read or write the state of all digital I/O in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. A read of an output returns the current logic level on the terminal, not necessarily the output state written. Writes are filtered by the value in DIO_INHIBIT.

Register: DIO_STATE

Address: 2800

```
property eio0: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: EIO0 (EIO#(0:7)@0)

Address: 2008

```
property eio1: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: EIO1 (EIO#(0:7)@1)

Address: 2009

```
property eio2: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: EIO2 (EIO#(0:7)@2)

Address: 2010

```
property eio3: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: EIO3 (EIO#(0:7)@3)

Address: 2011

```
property eio4: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: EIO4 (EIO#(0:7)@4)

Address: 2012

```
property eio5: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: EIO5 (EIO#(0:7)@5)

Address: 2013

```
property eio6: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: EIO6 (EIO#(0:7)@6)

Address: 2014

```
property eio7: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: EIO7 (EIO#(0:7)@7)

Address: 2015

```
property eio_cio_state: int
```

Read or write the state of the 12 bits of EIO-CIO in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. Reading lines set to output returns the current logic levels on the terminals, not necessarily the output states written. As of firmware 1.0172, MIO states are included in the upper nibble of the CIO byte.

Register: EIO_CIO_STATE

Address: 2581

```
property eio_direction: int
```

Read or write the direction of the 8 bits of EIO in a single binary-encoded value. 0=Input and 1=Output. The upper 8-bits of this value are inhibits.

Register: EIO_DIRECTION

Address: 2601

```
property eio_state: int
```

Read or write the state of the 8 bits of EIO in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. Reading lines set to output returns the current logic levels on the terminals, not necessarily the output states written. The upper 8-bits of this value are inhibits.

Register: EIO_STATE

Address: 2501

```
property ethernet_altdns: str
```

Read the current Alt DNS of wired Ethernet.

Register: ETHERNET_ALTDNS

Address: 49108

```
property ethernet_altdns_default: str
```

The Alt DNS of wired Ethernet after a power-cycle to the device.

Register: ETHERNET_ALTDNS_DEFAULT

Address: 49158

```
property ethernet_apply_settings: int
```

Writing 1 to this register power-cycles Ethernet. It tells the device to waits 1s before turning off Ethernet and then 500ms before turning it back on.

Register: ETHERNET_APPLY_SETTINGS

Address: 49190

```
property ethernet_dhcp_enable: int
```

Read the current Enabled/Disabled state of Ethernet DHCP.

Register: ETHERNET_DHCP_ENABLE

Address: 49110

```
property ethernet_dhcp_enable_default: int
```

The Enabled/Disabled state of Ethernet DHCP after a power-cycle to the device.

Register: ETHERNET_DHCP_ENABLE_DEFAULT

Address: 49160

```
property ethernet_dns: str
```

Read the current DNS of wired Ethernet.

Register: ETHERNET_DNS

Address: 49106

```
property ethernet_dns_default: str
```

The DNS of wired Ethernet after a power-cycle to the device.

Register: ETHERNET_DNS_DEFAULT

Address: 49156

```
property ethernet_gateway: str
```

Read the current gateway of wired Ethernet.

Register: ETHERNET_GATEWAY

Address: 49104

```
property ethernet_gateway_default: str
```

The gateway of wired Ethernet after a power-cycle to the device.

Register: ETHERNET_GATEWAY_DEFAULT

Address: 49154

```
property ethernet_ip: str
```

Read the current IP address of wired Ethernet.

Register: ETHERNET_IP

Address: 49100

```
property ethernet_ip_default: str
```

The IP address of wired Ethernet after a power-cycle to the device.

Register: ETHERNET_IP_DEFAULT

Address: 49150

```
property ethernet_mac: str
```

The MAC address of the wired Ethernet module.

Register: ETHERNET_MAC

Address: 60020

```
property ethernet_subnet: str
```

Read the current subnet of wired Ethernet.

Register: ETHERNET_SUBNET

Address: 49102

```
property ethernet_subnet_default: str
```

The subnet of wired Ethernet after a power-cycle to the device.

Register: ETHERNET_SUBNET_DEFAULT

Address: 49152

```
property ethernet_udp_discovery_only: int
```

Restricts which Modbus operations are allowed over UDP. When set to 1, only the registers needed for discovering units can be read and any other operation will throw an error.

Register: ETHERNET_UDP_DISCOVERY_ONLY

Address: 49115

```
property ethernet_udp_discovery_only_default: int
```

The Enabled/Disabled state of ETHERNET_UDP_DISCOVERY_ONLY after a power-cycle to the device.

Register: ETHERNET_UDP_DISCOVERY_ONLY_DEFAULT

Address: 49165

```
property file_io_attributes: int
```

Used to differentiate files from directories/folders. Bitmask: Bit0: Reserved, Bit1: Reserved, Bit2: Reserved, Bit3: Reserved, Bit4: 1=Directory, Bit5: 1=File.

Register: FILE_IO_ATTRIBUTES

Address: 60623

```
property file_io_close: int
```

Write any value to this register to close the open file.

Register: FILE_IO_CLOSE

Address: 60621

```
property file_io_delete: int
```

Write any value to this register to delete the active file. Must first designate which file to delete by writing to FILE_IO_PATH_WRITE_LEN_BYTES then FILE_IO_PATH_WRITE.

Register: FILE_IO_DELETE

Address: 60622

```
property file_io_dir_change: int
```

Write any value to this register to change the current working directory (CWD). Must first designate which directory to open by writing to FILE_IO_PATH_WRITE_LEN_BYTES then FILE_IO_PATH_WRITE.

Register: FILE_IO_DIR_CHANGE

Address: 60600

```
property file_io_dir_current: int
```

Write any value to this register to load the current working directory into FILE_IO_PATH_READ, and its length into FILE_IO_PATH_READ_LEN_BYTES. Can be used to identify current position in a file tree.

Register: FILE_IO_DIR_CURRENT

Address: 60601

```
property file_io_dir_first: int
```

Write any value to this register to initiate iteration through files and directories in the CWD. Typical sequence: FILE_IO_DIR_FIRST(W), then loop through: [FILE_IO_NAME_READ_LEN(R), FILE_IO_NAME_READ(R), FILE_IO_ATTRIBUTES(R), FILE_IO_SIZE_BYTES(R), FILE_IO_DIR_NEXT(W)] ..until any of the following errors: FILE_IO_END_OF_CWD (2966), FILE_IO_INVALID_OBJECT (2809), or FILE_IO_NOT_FOUND (2960).

Register: FILE_IO_DIR_FIRST

Address: 60610

```
property file_io_dir_make: int
```

Unimplemented.

Register: FILE_IO_DIR_MAKE

Address: 60602

```
property file_io_dir_next: int
```

Write any value to this register to continue iteration through files and directories in the CWD.

Register: FILE_IO_DIR_NEXT

Address: 60611

```
property file_io_dir_remove: int
```

Unimplemented.

Register: FILE_IO_DIR_REMOVE

Address: 60603

```
property file_io_disk_format_index: int
```

Used to determine the format of the SD card. 0=None or Unknown, 1=FAT12, 2=FAT16(Windows FAT), 3=FAT32.

Register: FILE_IO_DISK_FORMAT_INDEX

Address: 60638

```
property file_io_disk_free_clusters: int
```

Free (available) clusters in the SD card. Used to determine free space. Captured on read of FILE_IO_DISK_SECTOR_SIZE_BYTES.

Register: FILE_IO_DISK_FREE_CLUSTERS

Address: 60636

```
property file_io_disk_sector_size_bytes: int
```

The size of each sector in the SD card in bytes. In Windows this is called the Allocation Size.

Register: FILE_IO_DISK_SECTOR_SIZE_BYTES

Address: 60630

```
property file_io_disk_sectors_per_cluster: int
```

The number of sectors in each cluster. Captured on read of FILE_IO_DISK_SECTOR_SIZE_BYTES.

Register: FILE_IO_DISK_SECTORS_PER_CLUSTER

Address: 60632

```
property file_io_disk_total_clusters: int
```

The total number of clusters in the SD card. Captured on read of FILE_IO_DISK_SECTOR_SIZE_BYTES.

Register: FILE_IO_DISK_TOTAL_CLUSTERS

Address: 60634

```
property file_io_open: int
```

Write any value to this register to open a file. Must first designate which file to open by writing to FILE_IO_PATH_WRITE_LEN_BYTES then FILE_IO_PATH_WRITE.

Register: FILE_IO_OPEN

Address: 60620

```
property file_io_path_read: int
```

Read the next file path in the CWD. Length of the string (in bytes) determined by FILE_IO_PATH_READ_LEN_BYTES. File paths will be null terminated.

This register is a buffer. Underrun behavior - fill with zeros.

Register: FILE_IO_PATH_READ

Address: 60652

```
property file_io_path_read_len_bytes: int
```

Read the length (in bytes) of the next file path or directory to access.

Register: FILE_IO_PATH_READ_LEN_BYTES

Address: 60642

```
property file_io_path_write: int
```

Write the desired file path. Must first write the length of the file path string (in bytes) to FILE_IO_PATH_WRITE_LEN_BYTES. File paths should be null terminated. This register is a buffer.

Register: FILE_IO_PATH_WRITE

Address: 60650

```
property file_io_path_write_len_bytes: int
```

Write the length (in bytes) of the file path or directory to access.

Register: FILE_IO_PATH_WRITE_LEN_BYTES

Address: 60640

```
property file_io_read: int
```

Read the contents of a file. Must first write to FILE_IO_OPEN. Size of the file (in bytes) determined by FILE_IO_SIZE_BYTES.

This register is a buffer. Underrun behavior - throws an error.

Register: FILE_IO_READ

Address: 60656

```
property file_io_size_bytes: int
```

The size of the file in bytes. Directories have 0 size.

Register: FILE_IO_SIZE_BYTES

Address: 60628

```
property file_io_write: int
```

Unimplemented. This register is a buffer.

Register: FILE_IO_WRITE

Address: 60654

```
property fio0: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: FIO0 (FIO#(0:7)@0)

Address: 2000

```
property fio1: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: FIO1 (FIO#(0:7)@1)

Address: 2001

```
property fio2: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: FIO2 (FIO#(0:7)@2)

Address: 2002

```
property fio3: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: FIO3 (FIO#(0:7)@3)

Address: 2003

```
property fio4: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: FIO4 (FIO#(0:7)@4)

Address: 2004

```
property fio5: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: FIO5 (FIO#(0:7)@5)

Address: 2005

```
property fio6: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: FIO6 (FIO#(0:7)@6)

Address: 2006

```
property fio7: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: FIO7 (FIO#(0:7)@7)

Address: 2007

```
property fio_direction: int
```

Read or write the direction of the 8 bits of FIO in a single binary-encoded value. 0=Input and 1=Output. The upper 8-bits of this value are inhibits.

Register: FIO_DIRECTION

Address: 2600

```
property fio_eio_state: int
```

Read or write the state of the 16 bits of FIO-EIO in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. Reading lines set to output returns the current logic levels on the terminals, not necessarily the output states written.

Register: FIO_EIO_STATE

Address: 2580

```
property fio_state: int
```

Read or write the state of the 8 bits of FIO in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. Reading lines set to output returns the current logic levels on the terminals, not necessarily the output states written. The upper 8-bits of this value are inhibits.

Register: FIO_STATE

Address: 2500

```
property firmware_version: float
```

The current firmware version installed on the main processor.

Register: FIRMWARE_VERSION

Address: 60004

```
property hardware_installed: int
```

Bitmask indicating installed hardware options. bit0: High Resolution ADC, bit1: WiFi, bit2: RTC, bit3: microSD.

Register: HARDWARE_INSTALLED

Address: 60010

```
property hardware_version: float
```

The hardware version of the device.

Register: HARDWARE_VERSION

Address: 60002

```
property i2c_acks: int
```

A binary encoded value (array of bits) used to observe ACKs from the slave device.

Register: I2C_ACKS

Address: 5114

```
property i2c_data_rx: int
```

Data that has been read from the I2C bus.

This register is a buffer. Underrun behavior - fill with zeros.

Register: I2C_DATA_RX

Address: 5160

```
property i2c_data_tx: int
```

Data that will be written to the I2C bus. This register is a buffer.

Register: I2C_DATA_TX

Address: 5120

```
property i2c_go: int
```

Writing to this register will instruct the LabJack to perform an I2C transaction.

Register: I2C_GO

Address: 5110

```
property i2c_num_bytes_rx: int
```

The number of data bytes to read. Zero is valid and will result in a write-only I2C operation.

Register: I2C_NUM_BYTES_RX

Address: 5109

```
property i2c_num_bytes_tx: int
```

The number of data bytes to transmit. Zero is valid and will result in a read-only I2C operation.

Register: I2C_NUM_BYTES_TX

Address: 5108

```
property i2c_options: int
```

Advanced. Controls details of the I2C protocol to improve device compatibility. bit 0: 1 = Reset the I2C bus before attempting communication. bit 1: 0 = Restarts will use a stop and a start, 1 = Restarts will not use a stop. bit 2: 1 = disable clock stretching.

Register: I2C_OPTIONS

Address: 5103

```
property i2c_scl_dionum: int
```

The number of the DIO line to be used as the I2C clock line. Ex: Writing 1 will force FIO1 to become the I2C-SCL line.

Register: I2C_SCL_DIONUM

Address: 5101

```
property i2c_sda_dionum: int
```

The number of the DIO line to be used as the I2C data line. Ex: Writing 0 will force FIO0 to become the I2C-SDA line.

Register: I2C_SDA_DIONUM

Address: 5100

```
property i2c_slave_address: int
```

The 7-bit address of the slave device. Value is shifted left by firmware to allow room for the I2C R/W bit.

Register: I2C_SLAVE_ADDRESS

Address: 5104

```
property i2c_speed_throttle: int
```

This value controls the I2C clock frequency. Pass 0-65535. Default=0 corresponds to 65536 internally which results in ~450 kHz. 1 results in ~40 Hz, 65516 is ~100 kHz.

Register: I2C_SPEED_THROTTLE

Address: 5102

```
property internal_flash_calculate_crc: int
```

Calculates the checksum of a 4096 byte flash page.

Register: INTERNAL_FLASH_CALCULATE_CRC

Address: 61842

```
property internal_flash_erase: int
```

Erases a 4k section of internal flash starting at the specified address. This register is a buffer.

Register: INTERNAL_FLASH_ERASE

Address: 61820

```
property internal_flash_key: int
```

Sets the region of internal flash to which access is allowed.

Register: INTERNAL_FLASH_KEY

Address: 61800

```
property internal_flash_read: int
```

Data read from internal flash. Read size must be an even number of registers.

Register: INTERNAL_FLASH_READ

Address: 61812

```
property internal_flash_read_pointer: int
```

The address in internal flash that reads will start from.

Register: INTERNAL_FLASH_READ_POINTER

Address: 61810

```
property internal_flash_write: int
```

Data written here will be written to internal flash. Write size must be an even number of registers. This register is a buffer.

Register: INTERNAL_FLASH_WRITE

Address: 61832

```
property internal_flash_write_pointer: int
```

Address in internal flash where writes will begin.

Register: INTERNAL_FLASH_WRITE_POINTER

Address: 61830

```
property internal_is_amps: float
```

T8 Only. Returns the power supply current draw.

Register: INTERNAL_IS_AMPS

Address: 60032

```
property internal_vs_volts: float
```

T8 Only. Returns the power supply voltage.

Register: INTERNAL_VS_VOLTS

Address: 60030

```
property io_config_check_for_default: int
```

Returns 1 if the current configuration matches the default configuration.

Register: IO_CONFIG_CHECK_FOR_DEFAULT

Address: 49018

```
property io_config_check_for_factory: int
```

Returns 1 if default settings are the same as factory settings.

Register: IO_CONFIG_CHECK_FOR_FACTORY

Address: 49000

```
property io_config_current_crc32: int
```

Collects the current IO configuration and calculates of a CRC32.

Register: IO_CONFIG_CURRENT_CRC32

Address: 49020

```
property io_config_current_pread: int
```

The address in the current configuration data which reads will start from.

Register: IO_CONFIG_CURRENT_pREAD

Address: 49014

```
property io_config_current_read: int
```

Data read from the current configuration.

Register: IO_CONFIG_CURRENT_READ

Address: 49016

```
property io_config_default_pread: int
```

The address in the configuration data which reads will start from.

Register: IO_CONFIG_DEFAULT_pREAD

Address: 49010

```
property io_config_default_read: int
```

Data read from the startup configuration.

Register: IO_CONFIG_DEFAULT_READ

Address: 49012

```
property io_config_factory_pread: int
```

The address in the factory configuration data which reads will start from.

Register: IO_CONFIG_FACTORY_pREAD

Address: 49006

```
property io_config_factory_read: int
```

Data read from factory configuration.

Register: IO_CONFIG_FACTORY_READ

Address: 49008

```
property io_config_set_current_to_default: int
```

Write a 1 to set current values to default configuration. The default values are retrieved from flash and written to the current configuration registers, thus this behaves similar to reboot/power-up. Systems affected: AIN, DIO, DAC, AIN_EF, DIO_EF.

Register: IO_CONFIG_SET_CURRENT_TO_DEFAULT

Address: 61991

```
property io_config_set_current_to_factory: int
```

Write a 1 to set current values to factory configuration. The factory values are retrieved from flash and written to the current configuration registers. Systems affected: AIN, DIO, DAC, AIN_EF, DIO_EF.

Register: IO_CONFIG_SET_CURRENT_TO_FACTORY

Address: 61990

```
property io_config_set_default_to_current: int
```

Write a 1 to cause new default (reboot/power-up) values to be saved to flash. Current values are retrieved and saved as the new defaults. Systems affected: AIN, DIO, DAC, AIN_EF, DIO_EF.

Register: IO_CONFIG_SET_DEFAULT_TO_CURRENT

Address: 49002

```
property io_config_set_default_to_factory: int
```

Write a 1 to cause new default (reboot/power-up) values to be saved to flash. Factory values are retrieved and saved as the new defaults. Systems affected: AIN, DIO, DAC, AIN_EF, DIO_EF.

Register: IO_CONFIG_SET_DEFAULT_TO_FACTORY

Address: 49004

```
property last_err_detail: int
```

Returns the last LabJack error code seen on the device.

Register: LAST_ERR_DETAIL

Address: 55000

```
property last_err_frame: int
```

Returns the frame number that caused the last error.

Register: LAST_ERR_FRAME

Address: 55002

```
property last_err_transaction_id: int
```

Returns the transaction ID of the Modbus TCP packet that caused the last error.

Register: LAST_ERR_TRANSACTION_ID

Address: 55003

```
property last_mb_err: int
```

Returns the last Modbus TCP error code seen on the device.

Register: LAST_MB_ERR

Address: 55001

```
property led_comm: int
```

Sets the state of the COMM LED when the LEDs are set to manual, see the POWER_LED register.

Register: LED_COMM

Address: 2990

```
property led_status: int
```

Sets the state of the STATUS LED when the LEDs are set to manual, see the POWER_LED register.

Register: LED_STATUS

Address: 2991

```
property ljmcommon_version: float
```

The MAC address of the wired Ethernet module.

Register: LJMCOMMON_VERSION

Address: 60018

```
property lua_debug_data: int
```

Read debug data from here. This register is a buffer.

Register: LUA_DEBUG_DATA

Address: 6024

```
property lua_debug_enable: int
```

Write 1 to this register to enable debugging.

Register: LUA_DEBUG_ENABLE

Address: 6020

```
property lua_debug_enable_default: int
```

This is the value that will be loaded into LUA_DEBUG_ENABLE when the LabJack boots up.

Register: LUA_DEBUG_ENABLE_DEFAULT

Address: 6120

```
property lua_debug_num_bytes: int
```

The number of data bytes in the debug buffer waiting to be read.

Register: LUA_DEBUG_NUM_BYTES

Address: 6022

```
property lua_debug_num_bytes_default: int
```

This is the number of bytes that will be used for the lua debug buffer.

Register: LUA_DEBUG_NUM_BYTES_DEFAULT

Address: 6122

```
property lua_load_saved: int
```

Writing any value reads the script from Flash and loads it into RAM. The script can now be compiled and run.

Register: LUA_LOAD_SAVED

Address: 6034

```
property lua_no_warn_truncation: int
```

Writing a 1 to this register will prevent truncation warnings from being displayed. This register will be cleared every time Lua is started.

Register: LUA_NO_WARN_TRUNCATION

Address: 6050

```
property lua_run: int
```

Writing 1 compiles and runs the Lua script that is loaded in RAM. Writing zero stops the script and begins cleaning up memory. Users may poll the register after writing a value of 0 to verify that the VM is unloaded, and garbage collection is complete. 0 = VM fully unloaded. 1 = Running/in-progress.

Register: LUA_RUN

Address: 6000

```
property lua_run_default: int
```

If set to 1 the script saved in flash will be loaded and run when the LabJack boots up.

Register: LUA_RUN_DEFAULT

Address: 6100

```
property lua_save_to_flash: int
```

Write 1 to save the loaded source code to flash.

Register: LUA_SAVE_TO_FLASH

Address: 6032

```
property lua_saved_read: int
```

Read script saved to flash from here.

Register: LUA_SAVED_READ

Address: 6038

```
property lua_saved_read_pointer: int
```

Address within the saved Lua script in Flash to begin reading from.

Register: LUA_SAVED_READ_POINTER

Address: 6036

```
property lua_source_size: int
```

Allocates RAM for the source code.

Register: LUA_SOURCE_SIZE

Address: 6012

```
property lua_source_write: int
```

Write the source code here. Source will be saved to the RAM allocated by LUA_SOURCE_SIZE. This register is a buffer.

Register: LUA_SOURCE_WRITE

Address: 6014

```
property mio0: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: MIO0 (MIO#(0:2)@0)

Address: 2020

```
property mio1: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: MIO1 (MIO#(0:2)@1)

Address: 2021

```
property mio2: int
```

Read or set the state of 1 bit of digital I/O. Also configures the direction to input or output. Read 0=Low AND 1=High. Write 0=Low AND 1=High.

Register: MIO2 (MIO#(0:2)@2)

Address: 2022

```
property mio_direction: int
```

Read or write the direction of the 3 bits of MIO in a single binary-encoded value. 0=Input and 1=Output. The upper 8-bits of this value are inhibits.

Register: MIO_DIRECTION

Address: 2603

```
property mio_state: int
```

Read or write the state of the 3 bits of MIO in a single binary-encoded value. 0=Low AND 1=High. Does not configure direction. Reading lines set to output returns the current logic levels on the terminals, not necessarily the output states written. The upper 8-bits of this value are inhibits.

Register: MIO_STATE

Address: 2503

```
property onewire_data_rx: int
```

Data received over the 1-wire bus.

This register is a buffer. Underrun behavior - buffer is static, old data will fill the extra locations, firmware 1.0225 changes this to read zeros.

Register: ONEWIRE_DATA_RX

Address: 5370

```
property onewire_data_tx: int
```

Data to be transmitted over the 1-wire bus. This register is a buffer.

Register: ONEWIRE_DATA_TX

Address: 5340

```
property onewire_dpu_dionum: int
```

The dynamic pullup control DIO number.

Register: ONEWIRE_DPU_DIONUM

Address: 5301

```
property onewire_dq_dionum: int
```

The data-line DIO number.

Register: ONEWIRE_DQ_DIONUM

Address: 5300

```
property onewire_function: int
```

Set the ROM function to use. 0xF0=Search, 0xCC=Skip, 0x55=Match, 0x33=Read.

Register: ONEWIRE_FUNCTION

Address: 5307

```
property onewire_go: int
```

Instructs the device to perform the configured 1-wire transaction.

Register: ONEWIRE_GO

Address: 5310

```
property onewire_num_bytes_rx: int
```

Number of data bytes to be received.

Register: ONEWIRE_NUM_BYTES_RX

Address: 5309

```
property onewire_num_bytes_tx: int
```

Number of data bytes to be sent.

Register: ONEWIRE_NUM_BYTES_TX

Address: 5308

```
property onewire_options: int
```

Controls advanced features. Value is a bitmask. bit 0: reserved, bit 1: reserved, bit 2: 1=DPU Enabled 0=DPU Disabled, bit 3: DPU Polarity 1=Active state is high, 0=Active state is low (Dynamic Pull-Up)

Register: ONEWIRE_OPTIONS

Address: 5302

```
property onewire_path_h: int
```

Upper 32-bits of the path to take during a search.

Register: ONEWIRE_PATH_H

Address: 5324

```
property onewire_path_l: int
```

Lower 32-bits of the path to take during a search.

Register: ONEWIRE_PATH_L

Address: 5326

```
property onewire_rom_branchs_found_h: int
```

Upper 32-bits of the branches detected during a search.

Register: ONEWIRE_ROM_BRANCHS_FOUND_H

Address: 5332

```
property onewire_rom_branchs_found_l: int
```

Lower 32-bits of the branches detected during a search.

Register: ONEWIRE_ROM_BRANCHS_FOUND_L

Address: 5334

```
property onewire_rom_match_h: int
```

Upper 32-bits of the ROM to match.

Register: ONEWIRE_ROM_MATCH_H

Address: 5320

```
property onewire_rom_match_l: int
```

Lower 32-bits of the ROM to match.

Register: ONEWIRE_ROM_MATCH_L

Address: 5322

```
property onewire_search_result_h: int
```

Upper 32-bits of the search result.

Register: ONEWIRE_SEARCH_RESULT_H

Address: 5328

```
property onewire_search_result_l: int
```

Lower 32-bites of the search result.

Register: ONEWIRE_SEARCH_RESULT_L

Address: 5330

```
property power_ain: int
```

The current ON/OFF state of the analog input module.

Register: POWER_AIN

Address: 48005

```
property power_ain_default: int
```

The ON/OFF state of the analog input module after a power-cycle to the device. Provided to optionally reduce power consumption.

Register: POWER_AIN_DEFAULT

Address: 48055

```
property power_ethernet: int
```

The current ON/OFF state of the Ethernet module.

Register: POWER_ETHERNET

Address: 48003

```
property power_ethernet_default: int
```

The ON/OFF state of the Ethernet module after a power-cycle to the device. Provided to optionally reduce power consumption.

Register: POWER_ETHERNET_DEFAULT

Address: 48053

```
property power_led: int
```

Sets the LED operation: 0 = Off. Useful for lower power applications. 1 = normal. 2 = Lower power, LEDs will still blink but will normally be off. 3 = Reserved. 4 = Manual, in this mode the LEDs can be user controlled.

Register: POWER_LED

Address: 48006

```
property power_led_default: int
```

The ON/OFF state of the LEDs after a power-cycle to the device.

Register: POWER_LED_DEFAULT

Address: 48056

```
property power_wifi: int
```

The current ON/OFF state of the WiFi module.

Register: POWER_WIFI

Address: 48004

```
property power_wifi_default: int
```

The ON/OFF state of the WiFi module after a power-cycle to the device. Provided to optionally reduce power consumption.

Register: POWER_WIFI_DEFAULT

Address: 48054

```
property product_id: float
```

The numeric identifier of the device. Such as 7 for a T7 / T7-Pro.

Register: PRODUCT_ID

Address: 60000

```
property rtc_set_time_s: int
```

Write a new timestamp to the RTC in seconds since Jan, 1970, aka Epoch or Unix timestamp.

Register: RTC_SET_TIME_S

Address: 61504

```
property rtc_set_time_sntp: int
```

Write any value to instruct the T7 to update its clock from a SNTP server. Requires that SNTP_UPDATE_INTERVAL is non-zero.

Register: RTC_SET_TIME_SNTP

Address: 61506

```
property rtc_time_calendar: int
```

Read six consecutive addresses of type UINT16, starting with this address. The result will be in year, month, day, hour, minute, second calendar format. i.e. [2014, 10, 21, 18, 55, 35]

Register: RTC_TIME_CALENDAR

Address: 61510

```
property rtc_time_s: int
```

Read the current time in seconds since Jan, 1970, aka Epoch or Unix time. This value is calculated from the 80 MHz crystal, not the RTC 32 kHz crystal. Non-pro devices do not have a real time clock, so the reported time is either seconds since device startup or the time reported by the network time protocol over Ethernet. Pro devices have a real time clock that will be used to initialize this register at startup, the time can then be updated by NTP.

Register: RTC_TIME_S

Address: 61500

```
property sbus0_background_enable: int
```

Currently unsupported.

Register: SBUS0_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@0)

Address: 30250

```
property sbus0_clock_dionum: int
```

This is the DIO# that the external sensor’s clock line is connected to.

Register: SBUS0_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@0)

Address: 30225

```
property sbus0_data_dionum: int
```

This is the DIO# that the external sensor’s data line is connected to.

Register: SBUS0_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@0)

Address: 30200

```
property sbus0_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS0_RH (SBUS#(0:22)_RH@0)

Address: 30150

```
property sbus0_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS0_TEMP (SBUS#(0:22)_TEMP@0)

Address: 30100

```
property sbus10_background_enable: int
```

Currently unsupported.

Register: SBUS10_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@10)

Address: 30260

```
property sbus10_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS10_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@10)

Address: 30235

```
property sbus10_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS10_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@10)

Address: 30210

```
property sbus10_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS10_RH (SBUS#(0:22)_RH@10)

Address: 30170

```
property sbus10_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS10_TEMP (SBUS#(0:22)_TEMP@10)

Address: 30120

```
property sbus11_background_enable: int
```

Currently unsupported.

Register: SBUS11_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@11)

Address: 30261

```
property sbus11_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS11_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@11)

Address: 30236

```
property sbus11_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS11_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@11)

Address: 30211

```
property sbus11_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS11_RH (SBUS#(0:22)_RH@11)

Address: 30172

```
property sbus11_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS11_TEMP (SBUS#(0:22)_TEMP@11)

Address: 30122

```
property sbus12_background_enable: int
```

Currently unsupported.

Register: SBUS12_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@12)

Address: 30262

```
property sbus12_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS12_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@12)

Address: 30237

```
property sbus12_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS12_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@12)

Address: 30212

```
property sbus12_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS12_RH (SBUS#(0:22)_RH@12)

Address: 30174

```
property sbus12_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS12_TEMP (SBUS#(0:22)_TEMP@12)

Address: 30124

```
property sbus13_background_enable: int
```

Currently unsupported.

Register: SBUS13_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@13)

Address: 30263

```
property sbus13_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS13_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@13)

Address: 30238

```
property sbus13_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS13_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@13)

Address: 30213

```
property sbus13_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS13_RH (SBUS#(0:22)_RH@13)

Address: 30176

```
property sbus13_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS13_TEMP (SBUS#(0:22)_TEMP@13)

Address: 30126

```
property sbus14_background_enable: int
```

Currently unsupported.

Register: SBUS14_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@14)

Address: 30264

```
property sbus14_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS14_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@14)

Address: 30239

```
property sbus14_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS14_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@14)

Address: 30214

```
property sbus14_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS14_RH (SBUS#(0:22)_RH@14)

Address: 30178

```
property sbus14_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS14_TEMP (SBUS#(0:22)_TEMP@14)

Address: 30128

```
property sbus15_background_enable: int
```

Currently unsupported.

Register: SBUS15_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@15)

Address: 30265

```
property sbus15_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS15_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@15)

Address: 30240

```
property sbus15_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS15_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@15)

Address: 30215

```
property sbus15_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS15_RH (SBUS#(0:22)_RH@15)

Address: 30180

```
property sbus15_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS15_TEMP (SBUS#(0:22)_TEMP@15)

Address: 30130

```
property sbus16_background_enable: int
```

Currently unsupported.

Register: SBUS16_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@16)

Address: 30266

```
property sbus16_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS16_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@16)

Address: 30241

```
property sbus16_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS16_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@16)

Address: 30216

```
property sbus16_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS16_RH (SBUS#(0:22)_RH@16)

Address: 30182

```
property sbus16_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS16_TEMP (SBUS#(0:22)_TEMP@16)

Address: 30132

```
property sbus17_background_enable: int
```

Currently unsupported.

Register: SBUS17_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@17)

Address: 30267

```
property sbus17_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS17_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@17)

Address: 30242

```
property sbus17_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS17_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@17)

Address: 30217

```
property sbus17_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS17_RH (SBUS#(0:22)_RH@17)

Address: 30184

```
property sbus17_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS17_TEMP (SBUS#(0:22)_TEMP@17)

Address: 30134

```
property sbus18_background_enable: int
```

Currently unsupported.

Register: SBUS18_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@18)

Address: 30268

```
property sbus18_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS18_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@18)

Address: 30243

```
property sbus18_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS18_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@18)

Address: 30218

```
property sbus18_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS18_RH (SBUS#(0:22)_RH@18)

Address: 30186

```
property sbus18_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS18_TEMP (SBUS#(0:22)_TEMP@18)

Address: 30136

```
property sbus19_background_enable: int
```

Currently unsupported.

Register: SBUS19_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@19)

Address: 30269

```
property sbus19_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS19_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@19)

Address: 30244

```
property sbus19_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS19_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@19)

Address: 30219

```
property sbus19_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS19_RH (SBUS#(0:22)_RH@19)

Address: 30188

```
property sbus19_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS19_TEMP (SBUS#(0:22)_TEMP@19)

Address: 30138

```
property sbus1_background_enable: int
```

Currently unsupported.

Register: SBUS1_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@1)

Address: 30251

```
property sbus1_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS1_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@1)

Address: 30226

```
property sbus1_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS1_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@1)

Address: 30201

```
property sbus1_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS1_RH (SBUS#(0:22)_RH@1)

Address: 30152

```
property sbus1_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS1_TEMP (SBUS#(0:22)_TEMP@1)

Address: 30102

```
property sbus20_background_enable: int
```

Currently unsupported.

Register: SBUS20_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@20)

Address: 30270

```
property sbus20_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS20_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@20)

Address: 30245

```
property sbus20_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS20_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@20)

Address: 30220

```
property sbus20_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS20_RH (SBUS#(0:22)_RH@20)

Address: 30190

```
property sbus20_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS20_TEMP (SBUS#(0:22)_TEMP@20)

Address: 30140

```
property sbus21_background_enable: int
```

Currently unsupported.

Register: SBUS21_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@21)

Address: 30271

```
property sbus21_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS21_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@21)

Address: 30246

```
property sbus21_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS21_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@21)

Address: 30221

```
property sbus21_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS21_RH (SBUS#(0:22)_RH@21)

Address: 30192

```
property sbus21_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS21_TEMP (SBUS#(0:22)_TEMP@21)

Address: 30142

```
property sbus22_background_enable: int
```

Currently unsupported.

Register: SBUS22_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@22)

Address: 30272

```
property sbus22_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS22_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@22)

Address: 30247

```
property sbus22_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS22_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@22)

Address: 30222

```
property sbus22_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS22_RH (SBUS#(0:22)_RH@22)

Address: 30194

```
property sbus22_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS22_TEMP (SBUS#(0:22)_TEMP@22)

Address: 30144

```
property sbus2_background_enable: int
```

Currently unsupported.

Register: SBUS2_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@2)

Address: 30252

```
property sbus2_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS2_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@2)

Address: 30227

```
property sbus2_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS2_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@2)

Address: 30202

```
property sbus2_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS2_RH (SBUS#(0:22)_RH@2)

Address: 30154

```
property sbus2_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS2_TEMP (SBUS#(0:22)_TEMP@2)

Address: 30104

```
property sbus3_background_enable: int
```

Currently unsupported.

Register: SBUS3_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@3)

Address: 30253

```
property sbus3_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS3_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@3)

Address: 30228

```
property sbus3_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS3_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@3)

Address: 30203

```
property sbus3_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS3_RH (SBUS#(0:22)_RH@3)

Address: 30156

```
property sbus3_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS3_TEMP (SBUS#(0:22)_TEMP@3)

Address: 30106

```
property sbus4_background_enable: int
```

Currently unsupported.

Register: SBUS4_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@4)

Address: 30254

```
property sbus4_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS4_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@4)

Address: 30229

```
property sbus4_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS4_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@4)

Address: 30204

```
property sbus4_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS4_RH (SBUS#(0:22)_RH@4)

Address: 30158

```
property sbus4_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS4_TEMP (SBUS#(0:22)_TEMP@4)

Address: 30108

```
property sbus5_background_enable: int
```

Currently unsupported.

Register: SBUS5_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@5)

Address: 30255

```
property sbus5_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS5_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@5)

Address: 30230

```
property sbus5_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS5_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@5)

Address: 30205

```
property sbus5_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS5_RH (SBUS#(0:22)_RH@5)

Address: 30160

```
property sbus5_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS5_TEMP (SBUS#(0:22)_TEMP@5)

Address: 30110

```
property sbus6_background_enable: int
```

Currently unsupported.

Register: SBUS6_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@6)

Address: 30256

```
property sbus6_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS6_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@6)

Address: 30231

```
property sbus6_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS6_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@6)

Address: 30206

```
property sbus6_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS6_RH (SBUS#(0:22)_RH@6)

Address: 30162

```
property sbus6_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS6_TEMP (SBUS#(0:22)_TEMP@6)

Address: 30112

```
property sbus7_background_enable: int
```

Currently unsupported.

Register: SBUS7_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@7)

Address: 30257

```
property sbus7_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS7_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@7)

Address: 30232

```
property sbus7_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS7_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@7)

Address: 30207

```
property sbus7_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS7_RH (SBUS#(0:22)_RH@7)

Address: 30164

```
property sbus7_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS7_TEMP (SBUS#(0:22)_TEMP@7)

Address: 30114

```
property sbus8_background_enable: int
```

Currently unsupported.

Register: SBUS8_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@8)

Address: 30258

```
property sbus8_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS8_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@8)

Address: 30233

```
property sbus8_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS8_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@8)

Address: 30208

```
property sbus8_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS8_RH (SBUS#(0:22)_RH@8)

Address: 30166

```
property sbus8_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS8_TEMP (SBUS#(0:22)_TEMP@8)

Address: 30116

```
property sbus9_background_enable: int
```

Currently unsupported.

Register: SBUS9_BACKGROUND_ENABLE (SBUS#(0:22)_BACKGROUND_ENABLE@9)

Address: 30259

```
property sbus9_clock_dionum: int
```

This is the DIO# that the external sensor's clock line is connected to.

Register: SBUS9_CLOCK_DIONUM (SBUS#(0:22)_CLOCK_DIONUM@9)

Address: 30234

```
property sbus9_data_dionum: int
```

This is the DIO# that the external sensor's data line is connected to.

Register: SBUS9_DATA_DIONUM (SBUS#(0:22)_DATA_DIONUM@9)

Address: 30209

```
property sbus9_rh: float
```

Reads humidity in % from an external SBUS sensor (EI-1050/SHT1x/SHT7x). # is the DIO line for the EI-1050 enable. If # is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS9_RH (SBUS#(0:22)_RH@9)

Address: 30168

```
property sbus9_temp: float
```

Reads temperature in Kelvin from an SBUS sensor (EI-1050/SHT1x/SHT7x). SBUS# is the DIO line for the EI-1050 enable. If SBUS# is the same as the value specified for data or clock line, there will be no control of an enable line.

Register: SBUS9_TEMP (SBUS#(0:22)_TEMP@9)

Address: 30118

```
property sbus_all_clock_dionum: int
```

A write to this global parameter sets all SBUS clock line registers to the same value. A read will return the correct setting if all channels are set the same, but otherwise will return 0xFF.

Register: SBUS_ALL_CLOCK_DIONUM

Address: 30276

```
property sbus_all_clock_speed: int
```

Sets the clock speed. The clock is software generated so the resulting frequency is not exact. Valid range is 0-65535. Larger values are faster. 0 is the fastest option and is equivalent to 65536. A value of 0 is ~200 kHz. A value of 65000 is ~9.1 kHz.

Register: SBUS_ALL_CLOCK_SPEED

Address: 30278

```
property sbus_all_data_dionum: int
```

A write to this global parameter sets all SBUS data line registers to the same value. A read will return the correct setting if all channels are set the same, but otherwise will return 0xFF.

Register: SBUS_ALL_DATA_DIONUM

Address: 30275

```
property sbus_all_power_dionum: int
```

Sets the power line. This DIO is set to output-high upon any read of SBUS#_TEMP or SBUS#_RH. Default is FIO6 for the T4 and FIO2 for the T7. An FIO line can power up to 4 sensors while an EIO/CIO/MIO line or DAC line can power up to 20 sensors. Set to 9999 to disable. To use multiple power lines, use a DAC line for power, or otherwise control power yourself, set this to 9999 and then control power using writes to normal registers such as FIO5, EIO0, or DAC0.

Register: SBUS_ALL_POWER_DIONUM

Address: 30277

```
property serial_number: int
```

The serial number of the device.

Register: SERIAL_NUMBER

Address: 60028

```
setup_ain0(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN0.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain1(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN1.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain2(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN2.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain3(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN3.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain4(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN4.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain5(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN5.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain6(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN6.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain7(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0) →
None
```

Setup AIN7.

Parameters:

- **range** – the maximum expected voltage to automatically select the best gain
- **settling_time_us** – the settling time in micro seconds
- **resolution_index** – see LabJack docs

```
setup_ain_all(range_: float = 0.0, settling_time_us: int = 0, resolution_index: int = 0)
→ None
```

Setup all AIN channels.

```
property sntp_update_interval: int
```

Sets the SNTP retry time in seconds. A value of zero will disable SNTP(0=default). Values must be 10 or greater.

Register: SNTP_UPDATE_INTERVAL

Address: 49702

```
property spc_frequency_out_enable_deprecated: int
```

This has been deprecated in favor of DAC1_FREQUENCY_OUT_ENABLE. 0 = off, 1 = output 10 Hz signal on SPC. Note that stream uses SPC for a timing output, so if you use this while streaming you will get a lot more counts.

Register: SPC_FREQUENCY_OUT_ENABLE_DEPRECATED

Address: 61530

```
property spi_clk_dionum: int
```

The DIO line for Clock.

Register: SPI_CLK_DIONUM

Address: 5001

```
property spi_cs_dionum: int
```

The DIO line for Chip-Select.

Register: SPI_CS_DIONUM

Address: 5000

```
property spi_data_rx: int
```

Read data here.

This register is a buffer. Underrun behavior - fill with zeros.

Register: SPI_DATA_RX

Address: 5050

```
property spi_data_tx: int
```

Write data here. This register is a buffer.

Register: SPI_DATA_TX

Address: 5010

```
property spi_go: int
```

Write 1 to begin the configured SPI transaction.

Register: SPI_GO

Address: 5007

```
property spi_miso_dionum: int
```

The DIO line for Master-In-Slave-Out.

Register: SPI_MISO_DIONUM

Address: 5002

```
property spi_mode: int
```

The SPI mode controls the clock idle state and which edge clocks the data. Bit 1 is CPOL and Bit 0 is CPHA, so CPOL/CPHA for different decimal values: 0 = 0/0 = b00, 1 = 0/1 = b01, 2 = 1/0 = b10, 3 = 1/1 = b11. For CPOL and CPHA explanations, see Wikipedia article: https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus.

Register: SPI_MODE

Address: 5004

```
property spi_mosi_dionum: int
```

The DIO line for Master-Out-Slave-In.

Register: SPI_MOSI_DIONUM

Address: 5003

```
property spi_num_bytes: int
```

The number of bytes to transfer. The maximum transfer size is 100 bytes.

Register: SPI_NUM_BYTES

Address: 5009

```
property spi_options: int
```

Bit 0 is Auto-CS-Disable. When bit 0 is 0, CS is enabled. When bit 0 is 1, CS is disabled. Bit 1: 0 = Set DIO directions before starting the SPI operations, 1 = Do not set DIO directions. Bit 2: 0 = Transmit data MSB first, 1 = LSB first. Bits 4-7: This value sets the number of bits that will be transmitted during the last byte of the SPI operation. Default is 8, valid options are 1-8.

Register: SPI_OPTIONS

Address: 5006

```
property spi_speed_throttle: int
```

This value controls the SPI clock frequency. Pass 0-65535. Default=0 corresponds to 65536 internally which results in ~800 kHz. 65500 = ~100 kHz, 65100 = ~10 kHz, 61100 = ~1 kHz, 21000 = ~100 Hz, and 1 = ~67 Hz. Avoid setting too low such that the entire transaction lasts longer than the 250 millisecond timeout of the internal watchdog timer.

Register: SPI_SPEED_THROTTLE

Address: 5005

```
property stream_auto_target: int
```

Controls where data will be sent. Value is a bitmask. bit 0: 1 = Send to Ethernet 702 sockets, bit 1: 1 = Send to USB, bit 4: 1 = Command-Response mode. All other bits are reserved.

Register: STREAM_AUTO_TARGET

Address: 4016

```
property stream_autorecover_disable: int
```

When the stream buffer becomes full the stream will be stopped. Same behavior as if STREAM_ENABLE is set to zero.

Register: STREAM_AUTORECOVER_DISABLE

Address: 4028

```
property stream_buffer_size_bytes: int
```

Size of the stream data buffer in bytes. A value of 0 equates to the default value. Must be a power of 2. Size in samples is STREAM_BUFFER_SIZE_BYTES/2. Size in scans is (STREAM_BUFFER_SIZE_BYTES/2)/STREAM_NUM_ADDRESSES. Changes while stream is running do not affect the currently running stream.

Register: STREAM_BUFFER_SIZE_BYTES

Address: 4012

```
property stream_clock_source: int
```

Controls which clock source will be used to run the main stream clock. 0 = Internal crystal, 2 = External clock source on CIO3. Rising edges will increment a counter and trigger a stream scan after the number of edges specified in STREAM_EXTERNAL_CLOCK_DIVISOR. T7 will expect external stream clock on CIO3. All other values reserved.

Register: STREAM_CLOCK_SOURCE

Address: 4014

```
property stream_data_capture_16: int
```

If a channel produces 32-bits of data the upper 16 will be saved here.

Register: STREAM_DATA_CAPTURE_16

Address: 4899

```
property stream_data_cr: int
```

Address to read stream data from when operating in C-R mode.

Register: STREAM_DATA_CR

Address: 4500

```
property stream_debug_get_self_index: int
```

Returns the index of the channel. The index is the position in the scan list that the channel occupies.

Register: STREAM_DEBUG_GET_SELF_INDEX

Address: 4898

```
property stream_enable: int
```

Write 1 to start stream. Write 0 to stop stream. Reading this register returns 1 when stream is enabled. When using a triggered stream the stream is considered enabled while waiting for the trigger.

Register: STREAM_ENABLE

Address: 4990

```
property stream_external_clock_divisor: int
```

The number of pulses per stream scan when using an external clock.

Register: STREAM_EXTERNAL_CLOCK_DIVISOR

Address: 4022

```
property stream_num_addresses: int
```

The number of entries in the scanlist.

Register: STREAM_NUM_ADDRESSES

Address: 4004

```
property stream_num_scans: int
```

The number of scans to run before automatically stopping (stream-burst). 0 = run continuously. Limit for STREAM_NUM_SCANS is 2^32-1, but if the host is not reading data as fast as it is acquired you also need to consider STREAM_BUFFER_SIZE_BYTES.

Register: STREAM_NUM_SCANS

Address: 4020

```
property stream_out0: int
```

127) to trigger stream-out updates. When added to the scan list these do count against the max scan rate just like normal input addresses, but they do not return any data in the stream read.

Register: STREAM_OUT0 (STREAM_OUT#(0:3)@0)

Address: 4800

Type:
Include one or more of these registers in STREAM_SCANLIST_ADDRESS#(0

```
property stream_out0_buffer_allocate_num_bytes: int
```

Size of the buffer in bytes as a power of 2. Should be at least twice the size of updates that will be written and no less than 32. The usable buffer size will be equal to the value of this register minus 2 bytes (one 16-bit sample). Before writing data to _BUFFER_###, you must write to _BUFFER_ALLOCATE_NUM_BYTES to allocate RAM for the data. Max is 16384.

Register: STREAM_OUT0_BUFFER_ALLOCATE_NUM_BYTES (STREAM_OUT#(0:3)_BUFFER_ALLOCATE_NUM_BYTES@0)

Address: 4050

```
property stream_out0_buffer_f32: float
```

Data destination when sending floating point data. Appropriate cal constants are used to convert F32 values to 16-bit binary data, and thus each of these values uses 2 bytes of the stream-out buffer. This register is a buffer.

Register: STREAM_OUT0_BUFFER_F32 (STREAM_OUT#(0:3)_BUFFER_F32@0)

Address: 4400

```
property stream_out0_buffer_status: int
```

The number of values in the buffer that are not currently being used.

Register: STREAM_OUT0_BUFFER_STATUS (STREAM_OUT#(0:3)_BUFFER_STATUS@0)

Address: 4080

```
property stream_out0_buffer_u16: int
```

Data destination when sending 16-bit integer data. Each value uses 2 bytes of the stream-out buffer. This register is a buffer.

Register: STREAM_OUT0_BUFFER_U16 (STREAM_OUT#(0:3)_BUFFER_U16@0)

Address: 4420

```
property stream_out0_buffer_u32: int
```

Not used at this time. There are no U32 registers supported in stream-out. This register is a buffer.

Register: STREAM_OUT0_BUFFER_U32 (STREAM_OUT#(0:3)_BUFFER_U32@0)

Address: 4410

```
property stream_out0_enable: int
```

When STREAM_OUT#_ENABLE is enabled, the stream out target is generally updated by one value from the stream out buffer per stream scan. For example, there will generally be one instance of e.g. STREAM_OUT0 in the stream scan list, which will cause one STREAM_OUT0_BUFFER value to be consumed and written to STREAM_OUT0_TARGET for every stream scan. The stream scan list could also contain two instances of STREAM_OUT0, in which case two values from STREAM_OUT0_BUFFER value would be consumed and written for every stream scan.

Register: STREAM_OUT0_ENABLE (STREAM_OUT#(0:3)_ENABLE@0)

Address: 4090

```
property stream_out0_loop_num_values: int
```

The number of values, from the end of the array, that will be repeated after reaching the end of supplied data array.

Register: STREAM_OUT0_LOOP_NUM_VALUES (STREAM_OUT#(0:3)_LOOP_NUM_VALUES@0)

Address: 4060

```
property stream_out0_set_loop: int
```

Controls when new data and loop size are used. 1=Use new data immediately. 2=Wait for synch. New data will not be used until a different stream-out channel is set to Synch. 3=Synch. This stream-out# as well as any stream-outs set to synch will start using new data immediately.

Register: STREAM_OUT0_SET_LOOP (STREAM_OUT#(0:3)_SET_LOOP@0)

Address: 4070

```
property stream_out0_target: int
```

Channel that data will be written to. Before writing data to _BUFFER_###, you must write to _TARGET so the device knows how to interpret and store values.

Register: STREAM_OUT0_TARGET (STREAM_OUT#(0:3)_TARGET@0)

Address: 4040

```
property stream_out1: int
```

127) to trigger stream-out updates. When added to the scan list these do count against the max scan rate just like normal input addresses, but they do not return any data in the stream read.

Register: STREAM_OUT1 (STREAM_OUT#(0:3)@1)

Address: 4801

Type:
Include one or more of these registers in STREAM_SCANLIST_ADDRESS#(0

```
property stream_out1_buffer_allocate_num_bytes: int
```

Size of the buffer in bytes as a power of 2. Should be at least twice the size of updates that will be written and no less than 32. The usable buffer size will be equal to the value of this register minus 2 bytes (one 16-bit sample). Before writing data to _BUFFER_###, you must write to _BUFFER_ALLOCATE_NUM_BYTES to allocate RAM for the data. Max is 16384.

Register: STREAM_OUT1_BUFFER_ALLOCATE_NUM_BYTES (STREAM_OUT#(0:3)_BUFFER_ALLOCATE_NUM_BYTES@1)

Address: 4052

```
property stream_out1_buffer_f32: float
```

Data destination when sending floating point data. Appropriate cal constants are used to convert F32 values to 16-bit binary data, and thus each of these values uses 2 bytes of the stream-out buffer. This register is a buffer.

Register: STREAM_OUT1_BUFFER_F32 (STREAM_OUT#(0:3)_BUFFER_F32@1)

Address: 4402

```
property stream_out1_buffer_status: int
```

The number of values in the buffer that are not currently being used.

Register: STREAM_OUT1_BUFFER_STATUS (STREAM_OUT#(0:3)_BUFFER_STATUS@1)

Address: 4082

```
property stream_out1_buffer_u16: int
```

Data destination when sending 16-bit integer data. Each value uses 2 bytes of the stream-out buffer. This register is a buffer.

Register: STREAM_OUT1_BUFFER_U16 (STREAM_OUT#(0:3)_BUFFER_U16@1)

Address: 4421

```
property stream_out1_buffer_u32: int
```

Not used at this time. There are no U32 registers supported in stream-out. This register is a buffer.

Register: STREAM_OUT1_BUFFER_U32 (STREAM_OUT#(0:3)_BUFFER_U32@1)

Address: 4412

```
property stream_out1_enable: int
```

When STREAM_OUT#_ENABLE is enabled, the stream out target is generally updated by one value from the stream out buffer per stream scan. For example, there will generally be one instance of e.g. STREAM_OUT0 in the stream scan list, which will cause one STREAM_OUT0_BUFFER value to be consumed and written to STREAM_OUT0_TARGET for every stream scan. The stream scan list could also contain two instances of STREAM_OUT0, in which case two values from STREAM_OUT0_BUFFER value would be consumed and written for every stream scan.

Register: STREAM_OUT1_ENABLE (STREAM_OUT#(0:3)_ENABLE@1)

Address: 4092

```
property stream_out1_loop_num_values: int
```

The number of values, from the end of the array, that will be repeated after reaching the end of supplied data array.

Register: STREAM_OUT1_LOOP_NUM_VALUES (STREAM_OUT#(0:3)_LOOP_NUM_VALUES@1)

Address: 4062

```
property stream_out1_set_loop: int
```

Controls when new data and loop size are used. 1=Use new data immediately. 2=Wait for synch. New data will not be used until a different stream-out channel is set to Synch. 3=Synch. This stream-out# as well as any stream-outs set to synch will start using new data immediately.

Register: STREAM_OUT1_SET_LOOP (STREAM_OUT#(0:3)_SET_LOOP@1)

Address: 4072

```
property stream_out1_target: int
```

Channel that data will be written to. Before writing data to _BUFFER_###, you must write to _TARGET so the device knows how to interpret and store values.

Register: STREAM_OUT1_TARGET (STREAM_OUT#(0:3)_TARGET@1)

Address: 4042

```
property stream_out2: int
```

127) to trigger stream-out updates. When added to the scan list these do count against the max scan rate just like normal input addresses, but they do not return any data in the stream read.

Register: STREAM_OUT2 (STREAM_OUT#(0:3)@2)

Address: 4802

Type:
Include one or more of these registers in STREAM_SCANLIST_ADDRESS#(0

```
property stream_out2_buffer_allocate_num_bytes: int
```

Size of the buffer in bytes as a power of 2. Should be at least twice the size of updates that will be written and no less than 32. The usable buffer size will be equal to the value of this register minus 2 bytes (one 16-bit sample). Before writing data to _BUFFER_###, you must write to _BUFFER_ALLOCATE_NUM_BYTES to allocate RAM for the data. Max is 16384.

Register: STREAM_OUT2_BUFFER_ALLOCATE_NUM_BYTES
(STREAM_OUT#(0:3)_BUFFER_ALLOCATE_NUM_BYTES@2)

Address: 4054

```
property stream_out2_buffer_f32: float
```

Data destination when sending floating point data. Appropriate cal constants are used to convert F32 values to 16-bit binary data, and thus each of these values uses 2 bytes of the stream-out buffer. This register is a buffer.

Register: STREAM_OUT2_BUFFER_F32 (STREAM_OUT#(0:3)_BUFFER_F32@2)

Address: 4404

```
property stream_out2_buffer_status: int
```

The number of values in the buffer that are not currently being used.

Register: STREAM_OUT2_BUFFER_STATUS (STREAM_OUT#(0:3)_BUFFER_STATUS@2)

Address: 4084

```
property stream_out2_buffer_u16: int
```

Data destination when sending 16-bit integer data. Each value uses 2 bytes of the stream-out buffer. This register is a buffer.

Register: STREAM_OUT2_BUFFER_U16 (STREAM_OUT#(0:3)_BUFFER_U16@2)

Address: 4422

```
property stream_out2_buffer_u32: int
```

Not used at this time. There are no U32 registers supported in stream-out. This register is a buffer.

Register: STREAM_OUT2_BUFFER_U32 (STREAM_OUT#(0:3)_BUFFER_U32@2)

Address: 4414

```
property stream_out2_enable: int
```

When STREAM_OUT#_ENABLE is enabled, the stream out target is generally updated by one value from the stream out buffer per stream scan. For example, there will generally be one instance of e.g. STREAM_OUT0 in the stream scan list, which will cause one STREAM_OUT0_BUFFER value to be consumed and written to STREAM_OUT0_TARGET for every stream scan. The stream scan list could also contain two instances of STREAM_OUT0, in which case two values from STREAM_OUT0_BUFFER value would be consumed and written for every stream scan.

Register: STREAM_OUT2_ENABLE (STREAM_OUT#(0:3)_ENABLE@2)

Address: 4094

```
property stream_out2_loop_num_values: int
```

The number of values, from the end of the array, that will be repeated after reaching the end of supplied data array.

Register: STREAM_OUT2_LOOP_NUM_VALUES (STREAM_OUT#(0:3)_LOOP_NUM_VALUES@2)

Address: 4064

```
property stream_out2_set_loop: int
```

Controls when new data and loop size are used. 1=Use new data immediately. 2=Wait for synch. New data will not be used until a different stream-out channel is set to Synch. 3=Synch. This stream-out# as well as any stream-outs set to synch will start using new data immediately.

Register: STREAM_OUT2_SET_LOOP (STREAM_OUT#(0:3)_SET_LOOP@2)

Address: 4074

```
property stream_out2_target: int
```

Channel that data will be written to. Before writing data to _BUFFER_###, you must write to _TARGET so the device knows how to interpret and store values.

Register: STREAM_OUT2_TARGET (STREAM_OUT#(0:3)_TARGET@2)

Address: 4044

```
property stream_out3: int
```

127) to trigger stream-out updates. When added to the scan list these do count against the max scan rate just like normal input addresses, but they do not return any data in the stream read.

Register: STREAM_OUT3 (STREAM_OUT#(0:3)@3)

Address: 4803

Type:
Include one or more of these registers in STREAM_SCANLIST_ADDRESS#(0

```
property stream_out3_buffer_allocate_num_bytes: int
```

Size of the buffer in bytes as a power of 2. Should be at least twice the size of updates that will be written and no less than 32. The usable buffer size will be equal to the value of this register minus 2 bytes (one 16-bit sample). Before writing data to _BUFFER_###, you must write to _BUFFER_ALLOCATE_NUM_BYTES to allocate RAM for the data. Max is 16384.

Register: STREAM_OUT3_BUFFER_ALLOCATE_NUM_BYTES (STREAM_OUT#(0:3)_BUFFER_ALLOCATE_NUM_BYTES@3)

Address: 4056

```
property stream_out3_buffer_f32: float
```

Data destination when sending floating point data. Appropriate cal constants are used to convert F32 values to 16-bit binary data, and thus each of these values uses 2 bytes of the stream-out buffer. This register is a buffer.

Register: STREAM_OUT3_BUFFER_F32 (STREAM_OUT#(0:3)_BUFFER_F32@3)

Address: 4406

```
property stream_out3_buffer_status: int
```

The number of values in the buffer that are not currently being used.

Register: STREAM_OUT3_BUFFER_STATUS (STREAM_OUT#(0:3)_BUFFER_STATUS@3)

Address: 4086

```
property stream_out3_buffer_u16: int
```

Data destination when sending 16-bit integer data. Each value uses 2 bytes of the stream-out buffer. This register is a buffer.

Register: STREAM_OUT3_BUFFER_U16 (STREAM_OUT#(0:3)_BUFFER_U16@3)

Address: 4423

```
property stream_out3_buffer_u32: int
```

Not used at this time. There are no U32 registers supported in stream-out. This register is a buffer.

Register: STREAM_OUT3_BUFFER_U32 (STREAM_OUT#(0:3)_BUFFER_U32@3)

Address: 4416

```
property stream_out3_enable: int
```

When STREAM_OUT#_ENABLE is enabled, the stream out target is generally updated by one value from the stream out buffer per stream scan. For example, there will generally be one instance of e.g. STREAM_OUT0 in the stream scan list, which will cause one STREAM_OUT0_BUFFER value to be consumed and written to STREAM_OUT0_TARGET for every stream scan. The stream scan list could also contain two instances of STREAM_OUT0, in which case two values from STREAM_OUT0_BUFFER value would be consumed and written for every stream scan.

Register: STREAM_OUT3_ENABLE (STREAM_OUT#(0:3)_ENABLE@3)

Address: 4096

```
property stream_out3_loop_num_values: int
```

The number of values, from the end of the array, that will be repeated after reaching the end of supplied data array.

Register: STREAM_OUT3_LOOP_NUM_VALUES (STREAM_OUT#(0:3)_LOOP_NUM_VALUES@3)

Address: 4066

```
property stream_out3_set_loop: int
```

Controls when new data and loop size are used. 1=Use new data immediately. 2=Wait for synch. New data will not be used until a different stream-out channel is set to Synch. 3=Synch. This stream-out# as well as any stream-outs set to synch will start using new data immediately.

Register: STREAM_OUT3_SET_LOOP (STREAM_OUT#(0:3)_SET_LOOP@3)

Address: 4076

```
property stream_out3_target: int
```

Channel that data will be written to. Before writing data to _BUFFER_###, you must write to _TARGET so the device knows how to interpret and store values.

Register: STREAM_OUT3_TARGET (STREAM_OUT#(0:3)_TARGET@3)

Address: 4046

```
property stream_resolution_index: int
```

The resolution index for stream readings. A larger resolution index generally results in lower noise and longer sample times.

Register: STREAM_RESOLUTION_INDEX

Address: 4010

```
property stream_samples_per_packet: int
```

Specifies the number of data points to be sent in the data packet. Only applies to spontaneous mode.

Register: STREAM_SAMPLES_PER_PACKET

Address: 4006

```
property stream_scanlist_address0: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS0 (STREAM_SCANLIST_ADDRESS#(0:127)@0)

Address: 4100

```
property stream_scanlist_address1: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS1 (STREAM_SCANLIST_ADDRESS#(0:127)@1)

Address: 4102

```
property stream_scanlist_address10: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS10 (STREAM_SCANLIST_ADDRESS#(0:127)@10)

Address: 4120

```
property stream_scanlist_address100: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS100 (STREAM_SCANLIST_ADDRESS#(0:127)@100)

Address: 4300

```
property stream_scanlist_address101: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS101 (STREAM_SCANLIST_ADDRESS#(0:127)@101)

Address: 4302

```
property stream_scanlist_address102: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS102 (STREAM_SCANLIST_ADDRESS#(0:127)@102)

Address: 4304

```
property stream_scanlist_address103: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS103 (STREAM_SCANLIST_ADDRESS#(0:127)@103)

Address: 4306

```
property stream_scanlist_address104: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS104 (STREAM_SCANLIST_ADDRESS#(0:127)@104)

Address: 4308

```
property stream_scanlist_address105: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS105 (STREAM_SCANLIST_ADDRESS#(0:127)@105)

Address: 4310

```
property stream_scanlist_address106: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS106 (STREAM_SCANLIST_ADDRESS#(0:127)@106)

Address: 4312

```
property stream_scanlist_address107: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS107 (STREAM_SCANLIST_ADDRESS#(0:127)@107)

Address: 4314

```
property stream_scanlist_address108: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS108 (STREAM_SCANLIST_ADDRESS#(0:127)@108)

Address: 4316

```
property stream_scanlist_address109: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS109 (STREAM_SCANLIST_ADDRESS#(0:127)@109)

Address: 4318

```
property stream_scanlist_address11: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS11 (STREAM_SCANLIST_ADDRESS#(0:127)@11)

Address: 4122

```
property stream_scanlist_address110: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS110 (STREAM_SCANLIST_ADDRESS#(0:127)@110)

Address: 4320

```
property stream_scanlist_address111: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS111 (STREAM_SCANLIST_ADDRESS#(0:127)@111)

Address: 4322

```
property stream_scanlist_address112: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS112 (STREAM_SCANLIST_ADDRESS#(0:127)@112)

Address: 4324

```
property stream_scanlist_address113: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS113 (STREAM_SCANLIST_ADDRESS#(0:127)@113)

Address: 4326

```
property stream_scanlist_address114: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS114 (STREAM_SCANLIST_ADDRESS#(0:127)@114)

Address: 4328

```
property stream_scanlist_address115: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS115 (STREAM_SCANLIST_ADDRESS#(0:127)@115)

Address: 4330

```
property stream_scanlist_address116: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS116 (STREAM_SCANLIST_ADDRESS#(0:127)@116)

Address: 4332

```
property stream_scanlist_address117: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS117 (STREAM_SCANLIST_ADDRESS#(0:127)@117)

Address: 4334

```
property stream_scanlist_address118: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS118 (STREAM_SCANLIST_ADDRESS#(0:127)@118)

Address: 4336

```
property stream_scanlist_address119: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS119 (STREAM_SCANLIST_ADDRESS#(0:127)@119)

Address: 4338

```
property stream_scanlist_address12: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS12 (STREAM_SCANLIST_ADDRESS#(0:127)@12)

Address: 4124

```
property stream_scanlist_address120: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS120 (STREAM_SCANLIST_ADDRESS#(0:127)@120)

Address: 4340

```
property stream_scanlist_address121: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS121 (STREAM_SCANLIST_ADDRESS#(0:127)@121)

Address: 4342

```
property stream_scanlist_address122: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS122 (STREAM_SCANLIST_ADDRESS#(0:127)@122)

Address: 4344

```
property stream_scanlist_address123: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS123 (STREAM_SCANLIST_ADDRESS#(0:127)@123)

Address: 4346

```
property stream_scanlist_address124: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS124 (STREAM_SCANLIST_ADDRESS#(0:127)@124)

Address: 4348

```
property stream_scanlist_address125: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS125 (STREAM_SCANLIST_ADDRESS#(0:127)@125)

Address: 4350

```
property stream_scanlist_address126: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS126 (STREAM_SCANLIST_ADDRESS#(0:127)@126)

Address: 4352

```
property stream_scanlist_address127: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS127 (STREAM_SCANLIST_ADDRESS#(0:127)@127)

Address: 4354

```
property stream_scanlist_address13: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS13 (STREAM_SCANLIST_ADDRESS#(0:127)@13)

Address: 4126

```
property stream_scanlist_address14: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS14 (STREAM_SCANLIST_ADDRESS#(0:127)@14)

Address: 4128

```
property stream_scanlist_address15: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS15 (STREAM_SCANLIST_ADDRESS#(0:127)@15)

Address: 4130

```
property stream_scanlist_address16: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS16 (STREAM_SCANLIST_ADDRESS#(0:127)@16)

Address: 4132

```
property stream_scanlist_address17: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS17 (STREAM_SCANLIST_ADDRESS#(0:127)@17)

Address: 4134

```
property stream_scanlist_address18: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS18 (STREAM_SCANLIST_ADDRESS#(0:127)@18)

Address: 4136

```
property stream_scanlist_address19: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS19 (STREAM_SCANLIST_ADDRESS#(0:127)@19)

Address: 4138

```
property stream_scanlist_address2: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS2 (STREAM_SCANLIST_ADDRESS#(0:127)@2)

Address: 4104

```
property stream_scanlist_address20: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS20 (STREAM_SCANLIST_ADDRESS#(0:127)@20)

Address: 4140

```
property stream_scanlist_address21: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS21 (STREAM_SCANLIST_ADDRESS#(0:127)@21)

Address: 4142

```
property stream_scanlist_address22: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS22 (STREAM_SCANLIST_ADDRESS#(0:127)@22)

Address: 4144

```
property stream_scanlist_address23: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS23 (STREAM_SCANLIST_ADDRESS#(0:127)@23)

Address: 4146

```
property stream_scanlist_address24: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS24 (STREAM_SCANLIST_ADDRESS#(0:127)@24)

Address: 4148

```
property stream_scanlist_address25: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS25 (STREAM_SCANLIST_ADDRESS#(0:127)@25)

Address: 4150

```
property stream_scanlist_address26: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS26 (STREAM_SCANLIST_ADDRESS#(0:127)@26)

Address: 4152

```
property stream_scanlist_address27: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS27 (STREAM_SCANLIST_ADDRESS#(0:127)@27)

Address: 4154

```
property stream_scanlist_address28: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS28 (STREAM_SCANLIST_ADDRESS#(0:127)@28)

Address: 4156

```
property stream_scanlist_address29: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS29 (STREAM_SCANLIST_ADDRESS#(0:127)@29)

Address: 4158

```
property stream_scanlist_address3: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS3 (STREAM_SCANLIST_ADDRESS#(0:127)@3)

Address: 4106

```
property stream_scanlist_address30: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS30 (STREAM_SCANLIST_ADDRESS#(0:127)@30)

Address: 4160

```
property stream_scanlist_address31: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS31 (STREAM_SCANLIST_ADDRESS#(0:127)@31)

Address: 4162

```
property stream_scanlist_address32: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS32 (STREAM_SCANLIST_ADDRESS#(0:127)@32)

Address: 4164

```
property stream_scanlist_address33: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS33 (STREAM_SCANLIST_ADDRESS#(0:127)@33)

Address: 4166

```
property stream_scanlist_address34: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS34 (STREAM_SCANLIST_ADDRESS#(0:127)@34)

Address: 4168

```
property stream_scanlist_address35: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS35 (STREAM_SCANLIST_ADDRESS#(0:127)@35)

Address: 4170

```
property stream_scanlist_address36: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS36 (STREAM_SCANLIST_ADDRESS#(0:127)@36)

Address: 4172

```
property stream_scanlist_address37: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS37 (STREAM_SCANLIST_ADDRESS#(0:127)@37)

Address: 4174

```
property stream_scanlist_address38: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS38 (STREAM_SCANLIST_ADDRESS#(0:127)@38)

Address: 4176

```
property stream_scanlist_address39: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS39 (STREAM_SCANLIST_ADDRESS#(0:127)@39)

Address: 4178

```
property stream_scanlist_address4: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS4 (STREAM_SCANLIST_ADDRESS#(0:127)@4)

Address: 4108

```
property stream_scanlist_address40: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS40 (STREAM_SCANLIST_ADDRESS#(0:127)@40)

Address: 4180

```
property stream_scanlist_address41: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS41 (STREAM_SCANLIST_ADDRESS#(0:127)@41)

Address: 4182

```
property stream_scanlist_address42: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS42 (STREAM_SCANLIST_ADDRESS#(0:127)@42)

Address: 4184

```
property stream_scanlist_address43: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS43 (STREAM_SCANLIST_ADDRESS#(0:127)@43)

Address: 4186

```
property stream_scanlist_address44: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS44 (STREAM_SCANLIST_ADDRESS#(0:127)@44)

Address: 4188

```
property stream_scanlist_address45: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS45 (STREAM_SCANLIST_ADDRESS#(0:127)@45)

Address: 4190

```
property stream_scanlist_address46: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS46 (STREAM_SCANLIST_ADDRESS#(0:127)@46)

Address: 4192

```
property stream_scanlist_address47: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS47 (STREAM_SCANLIST_ADDRESS#(0:127)@47)

Address: 4194

```
property stream_scanlist_address48: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS48 (STREAM_SCANLIST_ADDRESS#(0:127)@48)

Address: 4196

```
property stream_scanlist_address49: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS49 (STREAM_SCANLIST_ADDRESS#(0:127)@49)

Address: 4198

```
property stream_scanlist_address5: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS5 (STREAM_SCANLIST_ADDRESS#(0:127)@5)

Address: 4110

```
property stream_scanlist_address50: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS50 (STREAM_SCANLIST_ADDRESS#(0:127)@50)

Address: 4200

```
property stream_scanlist_address51: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS51 (STREAM_SCANLIST_ADDRESS#(0:127)@51)

Address: 4202

```
property stream_scanlist_address52: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS52 (STREAM_SCANLIST_ADDRESS#(0:127)@52)

Address: 4204

```
property stream_scanlist_address53: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS53 (STREAM_SCANLIST_ADDRESS#(0:127)@53)

Address: 4206

```
property stream_scanlist_address54: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS54 (STREAM_SCANLIST_ADDRESS#(0:127)@54)

Address: 4208

```
property stream_scanlist_address55: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS55 (STREAM_SCANLIST_ADDRESS#(0:127)@55)

Address: 4210

```
property stream_scanlist_address56: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS56 (STREAM_SCANLIST_ADDRESS#(0:127)@56)

Address: 4212

```
property stream_scanlist_address57: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS57 (STREAM_SCANLIST_ADDRESS#(0:127)@57)

Address: 4214

```
property stream_scanlist_address58: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS58 (STREAM_SCANLIST_ADDRESS#(0:127)@58)

Address: 4216

```
property stream_scanlist_address59: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS59 (STREAM_SCANLIST_ADDRESS#(0:127)@59)

Address: 4218

```
property stream_scanlist_address6: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS6 (STREAM_SCANLIST_ADDRESS#(0:127)@6)

Address: 4112

```
property stream_scanlist_address60: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS60 (STREAM_SCANLIST_ADDRESS#(0:127)@60)

Address: 4220

```
property stream_scanlist_address61: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS61 (STREAM_SCANLIST_ADDRESS#(0:127)@61)

Address: 4222

```
property stream_scanlist_address62: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS62 (STREAM_SCANLIST_ADDRESS#(0:127)@62)

Address: 4224

```
property stream_scanlist_address63: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS63 (STREAM_SCANLIST_ADDRESS#(0:127)@63)

Address: 4226

```
property stream_scanlist_address64: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS64 (STREAM_SCANLIST_ADDRESS#(0:127)@64)

Address: 4228

```
property stream_scanlist_address65: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS65 (STREAM_SCANLIST_ADDRESS#(0:127)@65)

Address: 4230

```
property stream_scanlist_address66: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS66 (STREAM_SCANLIST_ADDRESS#(0:127)@66)

Address: 4232

```
property stream_scanlist_address67: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS67 (STREAM_SCANLIST_ADDRESS#(0:127)@67)

Address: 4234

```
property stream_scanlist_address68: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS68 (STREAM_SCANLIST_ADDRESS#(0:127)@68)

Address: 4236

```
property stream_scanlist_address69: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS69 (STREAM_SCANLIST_ADDRESS#(0:127)@69)

Address: 4238

```
property stream_scanlist_address7: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS7 (STREAM_SCANLIST_ADDRESS#(0:127)@7)

Address: 4114

```
property stream_scanlist_address70: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS70 (STREAM_SCANLIST_ADDRESS#(0:127)@70)

Address: 4240

```
property stream_scanlist_address71: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS71 (STREAM_SCANLIST_ADDRESS#(0:127)@71)

Address: 4242

```
property stream_scanlist_address72: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS72 (STREAM_SCANLIST_ADDRESS#(0:127)@72)

Address: 4244

```
property stream_scanlist_address73: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS73 (STREAM_SCANLIST_ADDRESS#(0:127)@73)

Address: 4246

```
property stream_scanlist_address74: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS74 (STREAM_SCANLIST_ADDRESS#(0:127)@74)

Address: 4248

```
property stream_scanlist_address75: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS75 (STREAM_SCANLIST_ADDRESS#(0:127)@75)

Address: 4250

```
property stream_scanlist_address76: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS76 (STREAM_SCANLIST_ADDRESS#(0:127)@76)

Address: 4252

```
property stream_scanlist_address77: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS77 (STREAM_SCANLIST_ADDRESS#(0:127)@77)

Address: 4254

```
property stream_scanlist_address78: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS78 (STREAM_SCANLIST_ADDRESS#(0:127)@78)

Address: 4256

```
property stream_scanlist_address79: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS79 (STREAM_SCANLIST_ADDRESS#(0:127)@79)

Address: 4258

```
property stream_scanlist_address8: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS8 (STREAM_SCANLIST_ADDRESS#(0:127)@8)

Address: 4116

```
property stream_scanlist_address80: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS80 (STREAM_SCANLIST_ADDRESS#(0:127)@80)

Address: 4260

```
property stream_scanlist_address81: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS81 (STREAM_SCANLIST_ADDRESS#(0:127)@81)

Address: 4262

```
property stream_scanlist_address82: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS82 (STREAM_SCANLIST_ADDRESS#(0:127)@82)

Address: 4264

```
property stream_scanlist_address83: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS83 (STREAM_SCANLIST_ADDRESS#(0:127)@83)

Address: 4266

```
property stream_scanlist_address84: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS84 (STREAM_SCANLIST_ADDRESS#(0:127)@84)

Address: 4268

```
property stream_scanlist_address85: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS85 (STREAM_SCANLIST_ADDRESS#(0:127)@85)

Address: 4270

```
property stream_scanlist_address86: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS86 (STREAM_SCANLIST_ADDRESS#(0:127)@86)

Address: 4272

```
property stream_scanlist_address87: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS87 (STREAM_SCANLIST_ADDRESS#(0:127)@87)

Address: 4274

```
property stream_scanlist_address88: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS88 (STREAM_SCANLIST_ADDRESS#(0:127)@88)

Address: 4276

```
property stream_scanlist_address89: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS89 (STREAM_SCANLIST_ADDRESS#(0:127)@89)

Address: 4278

```
property stream_scanlist_address9: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS9 (STREAM_SCANLIST_ADDRESS#(0:127)@9)

Address: 4118

```
property stream_scanlist_address90: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS90 (STREAM_SCANLIST_ADDRESS#(0:127)@90)

Address: 4280

```
property stream_scanlist_address91: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS91 (STREAM_SCANLIST_ADDRESS#(0:127)@91)

Address: 4282

```
property stream_scanlist_address92: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS92 (STREAM_SCANLIST_ADDRESS#(0:127)@92)

Address: 4284

```
property stream_scanlist_address93: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS93 (STREAM_SCANLIST_ADDRESS#(0:127)@93)

Address: 4286

```
property stream_scanlist_address94: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS94 (STREAM_SCANLIST_ADDRESS#(0:127)@94)

Address: 4288

```
property stream_scanlist_address95: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS95 (STREAM_SCANLIST_ADDRESS#(0:127)@95)

Address: 4290

```
property stream_scanlist_address96: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS96 (STREAM_SCANLIST_ADDRESS#(0:127)@96)

Address: 4292

```
property stream_scanlist_address97: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS97 (STREAM_SCANLIST_ADDRESS#(0:127)@97)

Address: 4294

```
property stream_scanlist_address98: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS98 (STREAM_SCANLIST_ADDRESS#(0:127)@98)

Address: 4296

```
property stream_scanlist_address99: int
```

A list of addresses to read each scan. In the case of Stream-Out enabled, the list may also include something to write each scan.

Register: STREAM_SCANLIST_ADDRESS99 (STREAM_SCANLIST_ADDRESS#(0:127)@99)

Address: 4298

```
property stream_scanrate_hz: float
```

Write a value to specify the number of times per second that all channels in the stream scanlist will be read. Max stream speeds are based on Sample Rate which is NumChannels*ScanRate. Has no effect when using an external clock. A read of this register returns the actual scan rate, which can be slightly different due to rounding. For scan rates >152.588, the actual scan interval is multiples of 100 ns. Assuming a core clock of 80 MHz the internal roll value is (80M/(8*DesiredScanRate))-1 and the actual scan rate is then 80M/(8*(RollValue+1). For slower scan rates the scan interval resolution is changed to 1 us, 10 us, 100 us, or 1 ms as needed to achieve the longer intervals.

Register: STREAM_SCANRATE_HZ

Address: 4002

```
property stream_settling_us: float
```

Time in microseconds to allow signals to settle after switching the mux. Does not apply to the 1st channel in the scan list, as that settling is controlled by scan rate (the time from the last channel until the start of the next scan). Default=0. When set to less than 1, automatic settling will be used. The automatic settling behavior varies by device.

Register: STREAM_SETTLING_US

Address: 4008

```
property stream_start_time_stamp: int
```

This register stores the value of CORE_TIMER at the start of the first stream scan. Note that the first stream scan happens 1 scan period after STREAM_ENABLE is set to 1.

Register: STREAM_START_TIME_STAMP

Address: 4026

```
property stream_trigger_index: int
```

Controls when stream scanning will start. 0 = Start when stream is enabled, 2000 = Start when DIO_EF0 detects an edge, 2001 = Start when DIO_EF1 detects an edge. See the stream documentation for all supported values.

Register: STREAM_TRIGGER_INDEX

Address: 4024

```
property system_counter_10khz: int
```

A 10 kHz counter synchronized to RTC_TIME_S. This register can be appended to RTC_TIME_S as the decimal portion to get 0.1 ms resolution.

Register: SYSTEM_COUNTER_10KHZ

Address: 61502

```
property system_reboot: int
```

Issues a device reboot. Must write 0x4C4Axxxx, where xxxx is number of 50ms ticks to wait before reboot. To reboot immediately write 0x4C4A0000 (d1279918080).

Register: SYSTEM_REBOOT

Address: 61998

```
property system_timer_20hz: int
```

Internal 32-bit system timer running at 20Hz.

Register: SYSTEM_TIMER_20HZ

Address: 61522

```
property tdac0: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC0 (TDAC#(0:21)@0)

Address: 30000

```
property tdac1: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC1 (TDAC#(0:21)@1)

Address: 30002

```
property tdac10: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC10 (TDAC#(0:21)@10)

Address: 30020

```
property tdac11: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC11 (TDAC#(0:21)@11)

Address: 30022

```
property tdac12: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC12 (TDAC#(0:21)@12)

Address: 30024

```
property tdac13: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC13 (TDAC#(0:21)@13)

Address: 30026

```
property tdac14: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC14 (TDAC#(0:21)@14)

Address: 30028

```
property tdac15: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC15 (TDAC#(0:21)@15)

Address: 30030

```
property tdac16: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC16 (TDAC#(0:21)@16)

Address: 30032

```
property tdac17: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC17 (TDAC#(0:21)@17)

Address: 30034

```
property tdac18: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC18 (TDAC#(0:21)@18)

Address: 30036

```
property tdac19: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC19 (TDAC#(0:21)@19)

Address: 30038

```
property tdac2: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC2 (TDAC#(0:21)@2)

Address: 30004

```
property tdac20: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC20 (TDAC#(0:21)@20)

Address: 30040

```
property tdac21: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC21 (TDAC#(0:21)@21)

Address: 30042

```
property tdac3: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC3 (TDAC#(0:21)@3)

Address: 30006

```
property tdac4: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC4 (TDAC#(0:21)@4)

Address: 30008

```
property tdac5: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC5 (TDAC#(0:21)@5)

Address: 30010

```
property tdac6: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC6 (TDAC#(0:21)@6)

Address: 30012

```
property tdac7: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC7 (TDAC#(0:21)@7)

Address: 30014

```
property tdac8: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC8 (TDAC#(0:21)@8)

Address: 30016

```
property tdac9: float
```

Update a voltage output on a connected LJTick-DAC accessory. Even TDAC# = DACA, Odd TDAC# = DACB. For instance, if LJTick-DAC accessory is connected to FIO2/FIO3 block on main device, TDAC2 corresponds with DACA, and TDAC3 corresponds with DACB.

Register: TDAC9 (TDAC#(0:21)@9)

Address: 30018

```
property tdac_serial_number: int
```

Returns the serial number of an LJTick-DAC, and forces a re-read of the calibration constants. Which LJTDAC is determined by the last write to TDAC# ... whether it was successful or not.

Register: TDAC_SERIAL_NUMBER

Address: 55200

```
property tdac_speed_throttle: int
```

Sets the I2C clock speed that will be used when communicating with the TDAC. Default value is 65516. See I2C_SPEED_THROTTLE for more detail.

Register: TDAC_SPEED_THROTTLE

Address: 55202

```
property temperature0: float
```

T8 Only. Returns the temperature of the specified analog input. And saves AIN and temperature inputs for all channels.

Register: TEMPERATURE0 (TEMPERATURE#(0:7)@0)

Address: 600

```
property temperature0_binary: int
```

T8 Only. Returns the 24-bit binary representation of the voltage output of the temperature sensor for the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command- response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: TEMPERATURE0_BINARY (TEMPERATURE#(0:7)_BINARY@0)

Address: 50600

```
property temperature0_binary_capture: int
```

T8 Only. Returns the saved 24-bit binary representation of the voltage output of the temperature sensor for the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: TEMPERATURE0_BINARY_CAPTURE (TEMPERATURE#(0:7)_BINARY_CAPTURE@0)

Address: 50700

```
property temperature0_capture: float
```

T8 Only. Returns the saved temperature of the specified analog input.

Register: TEMPERATURE0_CAPTURE (TEMPERATURE#(0:7)_CAPTURE@0)

Address: 700

```
property temperature1: float
```

T8 Only. Returns the temperature of the specified analog input. And saves AIN and temperature inputs for all channels.

Register: TEMPERATURE1 (TEMPERATURE#(0:7)@1)

Address: 602

```
property temperature1_binary: int
```

T8 Only. Returns the 24-bit binary representation of the voltage output of the temperature sensor for the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command- response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: TEMPERATURE1_BINARY (TEMPERATURE#(0:7)_BINARY@1)

Address: 50602

```
property temperature1_binary_capture: int
```

T8 Only. Returns the saved 24-bit binary representation of the voltage output of the temperature sensor for the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: TEMPERATURE1_BINARY_CAPTURE (TEMPERATURE#(0:7)_BINARY_CAPTURE@1)

Address: 50702

```
property temperature1_capture: float
```

T8 Only. Returns the saved temperature of the specified analog input.

Register: TEMPERATURE1_CAPTURE (TEMPERATURE#(0:7)_CAPTURE@1)

Address: 702

```
property temperature2: float
```

T8 Only. Returns the temperature of the specified analog input. And saves AIN and temperature inputs for all channels.

Register: TEMPERATURE2 (TEMPERATURE#(0:7)@2)

Address: 604

```
property temperature2_binary: int
```

T8 Only. Returns the 24-bit binary representation of the voltage output of the temperature sensor for the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command- response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: TEMPERATURE2_BINARY (TEMPERATURE#(0:7)_BINARY@2)

Address: 50604

```
property temperature2_binary_capture: int
```

T8 Only. Returns the saved 24-bit binary representation of the voltage output of the temperature sensor for the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: TEMPERATURE2_BINARY_CAPTURE (TEMPERATURE#(0:7)_BINARY_CAPTURE@2)

Address: 50704

```
property temperature2_capture: float
```

T8 Only. Returns the saved temperature of the specified analog input.

Register: TEMPERATURE2_CAPTURE (TEMPERATURE#(0:7)_CAPTURE@2)

Address: 704

```
property temperature3: float
```

T8 Only. Returns the temperature of the specified analog input. And saves AIN and temperature inputs for all channels.

Register: TEMPERATURE3 (TEMPERATURE#(0:7)@3)

Address: 606

```
property temperature3_binary: int
```

T8 Only. Returns the 24-bit binary representation of the voltage output of the temperature sensor for the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command- response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: TEMPERATURE3_BINARY (TEMPERATURE#(0:7)_BINARY@3)

Address: 50606

```
property temperature3_binary_capture: int
```

T8 Only. Returns the saved 24-bit binary representation of the voltage output of the temperature sensor for the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: TEMPERATURE3_BINARY_CAPTURE (TEMPERATURE#(0:7)_BINARY_CAPTURE@3)

Address: 50706

```
property temperature3_capture: float
```

T8 Only. Returns the saved temperature of the specified analog input.

Register: TEMPERATURE3_CAPTURE (TEMPERATURE#(0:7)_CAPTURE@3)

Address: 706

```
property temperature4: float
```

T8 Only. Returns the temperature of the specified analog input. And saves AIN and temperature inputs for all channels.

Register: TEMPERATURE4 (TEMPERATURE#(0:7)@4)

Address: 608

```
property temperature4_binary: int
```

T8 Only. Returns the 24-bit binary representation of the voltage output of the temperature sensor for the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command- response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: TEMPERATURE4_BINARY (TEMPERATURE#(0:7)_BINARY@4)

Address: 50608

```
property temperature4_binary_capture: int
```

T8 Only. Returns the saved 24-bit binary representation of the voltage output of the temperature sensor for the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: TEMPERATURE4_BINARY_CAPTURE (TEMPERATURE#(0:7)_BINARY_CAPTURE@4)

Address: 50708

```
property temperature4_capture: float
```

T8 Only. Returns the saved temperature of the specified analog input.

Register: TEMPERATURE4_CAPTURE (TEMPERATURE#(0:7)_CAPTURE@4)

Address: 708

```
property temperature5: float
```

T8 Only. Returns the temperature of the specified analog input. And saves AIN and temperature inputs for all channels.

Register: TEMPERATURE5 (TEMPERATURE#(0:7)@5)

Address: 610

```
property temperature5_binary: int
```

T8 Only. Returns the 24-bit binary representation of the voltage output of the temperature sensor for the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command- response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: TEMPERATURE5_BINARY (TEMPERATURE#(0:7)_BINARY@5)

Address: 50610

```
property temperature5_binary_capture: int
```

T8 Only. Returns the saved 24-bit binary representation of the voltage output of the temperature sensor for the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: TEMPERATURE5_BINARY_CAPTURE (TEMPERATURE#(0:7)_BINARY_CAPTURE@5)

Address: 50710

```
property temperature5_capture: float
```

T8 Only. Returns the saved temperature of the specified analog input.

Register: TEMPERATURE5_CAPTURE (TEMPERATURE#(0:7)_CAPTURE@5)

Address: 710

```
property temperature6: float
```

T8 Only. Returns the temperature of the specified analog input. And saves AIN and temperature inputs for all channels.

Register: TEMPERATURE6 (TEMPERATURE#(0:7)@6)

Address: 612

```
property temperature6_binary: int
```

T8 Only. Returns the 24-bit binary representation of the voltage output of the temperature sensor for the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command- response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: TEMPERATURE6_BINARY (TEMPERATURE#(0:7)_BINARY@6)

Address: 50612

```
property temperature6_binary_capture: int
```

T8 Only. Returns the saved 24-bit binary representation of the voltage output of the temperature sensor for the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: TEMPERATURE6_BINARY_CAPTURE (TEMPERATURE#(0:7)_BINARY_CAPTURE@6)

Address: 50712

```
property temperature6_capture: float
```

T8 Only. Returns the saved temperature of the specified analog input.

Register: TEMPERATURE6_CAPTURE (TEMPERATURE#(0:7)_CAPTURE@6)

Address: 712

```
property temperature7: float
```

T8 Only. Returns the temperature of the specified analog input. And saves AIN and temperature inputs for all channels.

Register: TEMPERATURE7 (TEMPERATURE#(0:7)@7)

Address: 614

```
property temperature7_binary: int
```

T8 Only. Returns the 24-bit binary representation of the voltage output of the temperature sensor for the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command- response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: TEMPERATURE7_BINARY (TEMPERATURE#(0:7)_BINARY@7)

Address: 50614

```
property temperature7_binary_capture: int
```

T8 Only. Returns the saved 24-bit binary representation of the voltage output of the temperature sensor for the specified analog input. If you prefer 16-bit representation, simply divide this by 256. This is for command-response only. Stream always returns binary and LJM applies cal constants, so the LJM config flag LJM_STREAM_AIN_BINARY is used to get binary values.

Register: TEMPERATURE7_BINARY_CAPTURE (TEMPERATURE#(0:7)_BINARY_CAPTURE@7)

Address: 50714

```
property temperature7_capture: float
```

T8 Only. Returns the saved temperature of the specified analog input.

Register: TEMPERATURE7_CAPTURE (TEMPERATURE#(0:7)_CAPTURE@7)

Address: 714

```
property temperature_air_k: float
```

Returns the estimated ambient air temperature just outside of the device in its red plastic enclosure. This register is equal to TEMPERATURE_DEVICE_K - 4.3. If Ethernet and/or WiFi is enabled, subtract an extra 0.6 for each.

Register: TEMPERATURE_AIR_K

Address: 60050

```
property temperature_device_k: float
```

Takes a reading from the internal temperature sensor using range=+/-10V and resolution=8, and applies the formula Volts*-92.6+467.6 to return kelvins.

Register: TEMPERATURE_DEVICE_K

Address: 60052

```
property test: int
```

A read of this test register should always return 0x00112233 or d1122867. If your software has the word swap quirk, you will incorrectly read 0x22330011 or 573767697. If your software has the address-1 quirk, a UINT16 (1-register) read from 55101 will incorrectly return 0x0011 (should read 0x2233). This register is unlike others, in that it allows you can read a single word from 55100 or 55101, or of course 2 words from 55100.

Register: TEST

Address: 55100

```
property test_float32: float
```

Write a value and read back to test FLOAT32 operation. Default is 0xC61C3C00 or -9999.0. If your software has the word swap quirk, the default will incorrectly read 0x3C00C61C or 0.00786.

Register: TEST_FLOAT32

Address: 55124

```
property test_int32: int
```

Write a value and read back to test INT32 operation. Default is 0x8899AABB or d-2003195205. If your software has the word swap quirk, the default will incorrectly read 0xAABB8899 or -1430550375.

Register: TEST_INT32

Address: 55122

```
property test_uint16: int
```

Write a value and read back to test UINT16 operation. Default is 0x0011 or d17.

Register: TEST_UINT16

Address: 55110

```
property test_uint32: int
```

Write a value and read back to test UINT32 operation. Default is 0x00112233 or d1122867. If your software has the word swap quirk, the default will incorrectly read 0x22330011 or 573767697.

Register: TEST_UINT32

Address: 55120

```
property user_ram0_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM0_F32 (USER_RAM#(0:39)_F32@0)

Address: 46000

```
property user_ram0_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM0_I32 (USER_RAM#(0:9)_I32@0)

Address: 46080

```
property user_ram0_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM0_U16 (USER_RAM#(0:19)_U16@0)

Address: 46180

```
property user_ram0_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM0_U32 (USER_RAM#(0:39)_U32@0)

Address: 46100

```
property user_ram10_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM10_F32 (USER_RAM#(0:39)_F32@10)

Address: 46020

```
property user_ram10_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM10_U16 (USER_RAM#(0:19)_U16@10)

Address: 46190

```
property user_ram10_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM10_U32 (USER_RAM#(0:39)_U32@10)

Address: 46120

```
property user_ram11_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM11_F32 (USER_RAM#(0:39)_F32@11)

Address: 46022

```
property user_ram11_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM11_U16 (USER_RAM#(0:19)_U16@11)

Address: 46191

```
property user_ram11_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM11_U32 (USER_RAM#(0:39)_U32@11)

Address: 46122

```
property user_ram12_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM12_F32 (USER_RAM#(0:39)_F32@12)

Address: 46024

```
property user_ram12_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM12_U16 (USER_RAM#(0:19)_U16@12)

Address: 46192

```
property user_ram12_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM12_U32 (USER_RAM#(0:39)_U32@12)

Address: 46124

```
property user_ram13_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM13_F32 (USER_RAM#(0:39)_F32@13)

Address: 46026

```
property user_ram13_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM13_U16 (USER_RAM#(0:19)_U16@13)

Address: 46193

```
property user_ram13_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM13_U32 (USER_RAM#(0:39)_U32@13)

Address: 46126

```
property user_ram14_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM14_F32 (USER_RAM#(0:39)_F32@14)

Address: 46028

```
property user_ram14_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM14_U16 (USER_RAM#(0:19)_U16@14)

Address: 46194

```
property user_ram14_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM14_U32 (USER_RAM#(0:39)_U32@14)

Address: 46128

```
property user_ram15_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM15_F32 (USER_RAM#(0:39)_F32@15)

Address: 46030

```
property user_ram15_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM15_U16 (USER_RAM#(0:19)_U16@15)

Address: 46195

```
property user_ram15_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM15_U32 (USER_RAM#(0:39)_U32@15)

Address: 46130

```
property user_ram16_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM16_F32 (USER_RAM#(0:39)_F32@16)

Address: 46032

```
property user_ram16_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM16_U16 (USER_RAM#(0:19)_U16@16)

Address: 46196

```
property user_ram16_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM16_U32 (USER_RAM#(0:39)_U32@16)

Address: 46132

```
property user_ram17_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM17_F32 (USER_RAM#(0:39)_F32@17)

Address: 46034

```
property user_ram17_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM17_U16 (USER_RAM#(0:19)_U16@17)

Address: 46197

```
property user_ram17_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM17_U32 (USER_RAM#(0:39)_U32@17)

Address: 46134

```
property user_ram18_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM18_F32 (USER_RAM#(0:39)_F32@18)

Address: 46036

```
property user_ram18_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM18_U16 (USER_RAM#(0:19)_U16@18)

Address: 46198

```
property user_ram18_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM18_U32 (USER_RAM#(0:39)_U32@18)

Address: 46136

```
property user_ram19_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM19_F32 (USER_RAM#(0:39)_F32@19)

Address: 46038

```
property user_ram19_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM19_U16 (USER_RAM#(0:19)_U16@19)

Address: 46199

```
property user_ram19_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM19_U32 (USER_RAM#(0:39)_U32@19)

Address: 46138

```
property user_ram1_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM1_F32 (USER_RAM#(0:39)_F32@1)

Address: 46002

```
property user_ram1_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM1_I32 (USER_RAM#(0:9)_I32@1)

Address: 46082

```
property user_ram1_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM1_U16 (USER_RAM#(0:19)_U16@1)

Address: 46181

```
property user_ram1_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM1_U32 (USER_RAM#(0:39)_U32@1)

Address: 46102

```
property user_ram20_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM20_F32 (USER_RAM#(0:39)_F32@20)

Address: 46040

```
property user_ram20_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM20_U32 (USER_RAM#(0:39)_U32@20)

Address: 46140

```
property user_ram21_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM21_F32 (USER_RAM#(0:39)_F32@21)

Address: 46042

```
property user_ram21_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM21_U32 (USER_RAM#(0:39)_U32@21)

Address: 46142

```
property user_ram22_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM22_F32 (USER_RAM#(0:39)_F32@22)

Address: 46044

```
property user_ram22_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM22_U32 (USER_RAM#(0:39)_U32@22)

Address: 46144

```
property user_ram23_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM23_F32 (USER_RAM#(0:39)_F32@23)

Address: 46046

```
property user_ram23_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM23_U32 (USER_RAM#(0:39)_U32@23)

Address: 46146

```
property user_ram24_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM24_F32 (USER_RAM#(0:39)_F32@24)

Address: 46048

```
property user_ram24_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM24_U32 (USER_RAM#(0:39)_U32@24)

Address: 46148

```
property user_ram25_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM25_F32 (USER_RAM#(0:39)_F32@25)

Address: 46050

```
property user_ram25_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM25_U32 (USER_RAM#(0:39)_U32@25)

Address: 46150

```
property user_ram26_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM26_F32 (USER_RAM#(0:39)_F32@26)

Address: 46052

```
property user_ram26_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM26_U32 (USER_RAM#(0:39)_U32@26)

Address: 46152

```
property user_ram27_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM27_F32 (USER_RAM#(0:39)_F32@27)

Address: 46054

```
property user_ram27_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM27_U32 (USER_RAM#(0:39)_U32@27)

Address: 46154

```
property user_ram28_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM28_F32 (USER_RAM#(0:39)_F32@28)

Address: 46056

```
property user_ram28_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM28_U32 (USER_RAM#(0:39)_U32@28)

Address: 46156

```
property user_ram29_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM29_F32 (USER_RAM#(0:39)_F32@29)

Address: 46058

```
property user_ram29_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM29_U32 (USER_RAM#(0:39)_U32@29)

Address: 46158

```
property user_ram2_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM2_F32 (USER_RAM#(0:39)_F32@2)

Address: 46004

```
property user_ram2_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM2_I32 (USER_RAM#(0:9)_I32@2)

Address: 46084

```
property user_ram2_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM2_U16 (USER_RAM#(0:19)_U16@2)

Address: 46182

```
property user_ram2_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM2_U32 (USER_RAM#(0:39)_U32@2)

Address: 46104

```
property user_ram30_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM30_F32 (USER_RAM#(0:39)_F32@30)

Address: 46060

```
property user_ram30_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM30_U32 (USER_RAM#(0:39)_U32@30)

Address: 46160

```
property user_ram31_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM31_F32 (USER_RAM#(0:39)_F32@31)

Address: 46062

```
property user_ram31_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM31_U32 (USER_RAM#(0:39)_U32@31)

Address: 46162

```
property user_ram32_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM32_F32 (USER_RAM#(0:39)_F32@32)

Address: 46064

```
property user_ram32_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM32_U32 (USER_RAM#(0:39)_U32@32)

Address: 46164

```
property user_ram33_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM33_F32 (USER_RAM#(0:39)_F32@33)

Address: 46066

```
property user_ram33_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM33_U32 (USER_RAM#(0:39)_U32@33)

Address: 46166

```
property user_ram34_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM34_F32 (USER_RAM#(0:39)_F32@34)

Address: 46068

```
property user_ram34_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM34_U32 (USER_RAM#(0:39)_U32@34)

Address: 46168

```
property user_ram35_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM35_F32 (USER_RAM#(0:39)_F32@35)

Address: 46070

```
property user_ram35_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM35_U32 (USER_RAM#(0:39)_U32@35)

Address: 46170

```
property user_ram36_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM36_F32 (USER_RAM#(0:39)_F32@36)

Address: 46072

```
property user_ram36_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM36_U32 (USER_RAM#(0:39)_U32@36)

Address: 46172

```
property user_ram37_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM37_F32 (USER_RAM#(0:39)_F32@37)

Address: 46074

```
property user_ram37_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM37_U32 (USER_RAM#(0:39)_U32@37)

Address: 46174

```
property user_ram38_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM38_F32 (USER_RAM#(0:39)_F32@38)

Address: 46076

```
property user_ram38_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM38_U32 (USER_RAM#(0:39)_U32@38)

Address: 46176

```
property user_ram39_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM39_F32 (USER_RAM#(0:39)_F32@39)

Address: 46078

```
property user_ram39_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM39_U32 (USER_RAM#(0:39)_U32@39)

Address: 46178

```
property user_ram3_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM3_F32 (USER_RAM#(0:39)_F32@3)

Address: 46006

```
property user_ram3_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM3_I32 (USER_RAM#(0:9)_I32@3)

Address: 46086

```
property user_ram3_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM3_U16 (USER_RAM#(0:19)_U16@3)

Address: 46183

```
property user_ram3_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM3_U32 (USER_RAM#(0:39)_U32@3)

Address: 46106

```
property user_ram4_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM4_F32 (USER_RAM#(0:39)_F32@4)

Address: 46008

```
property user_ram4_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM4_I32 (USER_RAM#(0:9)_I32@4)

Address: 46088

```
property user_ram4_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM4_U16 (USER_RAM#(0:19)_U16@4)

Address: 46184

```
property user_ram4_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM4_U32 (USER_RAM#(0:39)_U32@4)

Address: 46108

```
property user_ram5_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM5_F32 (USER_RAM#(0:39)_F32@5)

Address: 46010

```
property user_ram5_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM5_I32 (USER_RAM#(0:9)_I32@5)

Address: 46090

```
property user_ram5_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM5_U16 (USER_RAM#(0:19)_U16@5)

Address: 46185

```
property user_ram5_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM5_U32 (USER_RAM#(0:39)_U32@5)

Address: 46110

```
property user_ram6_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM6_F32 (USER_RAM#(0:39)_F32@6)

Address: 46012

```
property user_ram6_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM6_I32 (USER_RAM#(0:9)_I32@6)

Address: 46092

```
property user_ram6_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM6_U16 (USER_RAM#(0:19)_U16@6)

Address: 46186

```
property user_ram6_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM6_U32 (USER_RAM#(0:39)_U32@6)

Address: 46112

```
property user_ram7_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM7_F32 (USER_RAM#(0:39)_F32@7)

Address: 46014

```
property user_ram7_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM7_I32 (USER_RAM#(0:9)_I32@7)

Address: 46094

```
property user_ram7_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM7_U16 (USER_RAM#(0:19)_U16@7)

Address: 46187

```
property user_ram7_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM7_U32 (USER_RAM#(0:39)_U32@7)

Address: 46114

```
property user_ram8_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM8_F32 (USER_RAM#(0:39)_F32@8)

Address: 46016

```
property user_ram8_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM8_I32 (USER_RAM#(0:9)_I32@8)

Address: 46096

```
property user_ram8_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM8_U16 (USER_RAM#(0:19)_U16@8)

Address: 46188

```
property user_ram8_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM8_U32 (USER_RAM#(0:39)_U32@8)

Address: 46116

```
property user_ram9_f32: float
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM9_F32 (USER_RAM#(0:39)_F32@9)

Address: 46018

```
property user_ram9_i32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM9_I32 (USER_RAM#(0:9)_I32@9)

Address: 46098

```
property user_ram9_u16: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM9_U16 (USER_RAM#(0:19)_U16@9)

Address: 46189

```
property user_ram9_u32: int
```

Generic RAM registers. Useful for passing data between a host computer and a Lua script. Will not return an error if alternate data types are used.

Register: USER_RAM9_U32 (USER_RAM#(0:39)_U32@9)

Address: 46118

```
property user_ram_fifo0_allocate_num_bytes: int
```

Allocate memory for a FIFO buffer. Number of bytes should be sufficient to store users max transfer array size. Note that FLOAT32, INT32, and UINT32 require 4 bytes per value, and UINT16 require 2 bytes per value. Maximum size is limited by available memory. Care should be taken to conserve enough memory for other operations such as AIN_EF, Lua, Stream etc.

Register: USER_RAM_FIFO0_ALLOCATE_NUM_BYTES
(USER_RAM_FIFO#(0:3)_ALLOCATE_NUM_BYTES@0)

Address: 47900

```
property user_ram_fifo0_data_f32: float
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO0_DATA_F32 (USER_RAM_FIFO#(0:3)_DATA_F32@0)

Address: 47030

```
property user_ram_fifo0_data_i32: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO0_DATA_I32 (USER_RAM_FIFO#(0:3)_DATA_I32@0)

Address: 47020

```
property user_ram_fifo0_data_u16: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO0_DATA_U16 (USER_RAM_FIFO#(0:3)_DATA_U16@0)

Address: 47000

```
property user_ram_fifo0_data_u32: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO0_DATA_U32 (USER_RAM_FIFO#(0:3)_DATA_U32@0)

Address: 47010

```
property user_ram_fifo0_empty: int
```

Write any value to this register to efficiently empty, flush, or otherwise clear data from the FIFO.

Register: USER_RAM_FIFO0_EMPTY (USER_RAM_FIFO#(0:3)_EMPTY@0)

Address: 47930

```
property user_ram_fifo0_num_bytes_in_fifo: int
```

Poll this register to see when new data is available/ready. Each read of the FIFO buffer decreases this value, and each write to the FIFO buffer increases this value.

At any point in time, the following equation holds: Nbytes = Nwritten - Nread.

Register: USER_RAM_FIFO0_NUM_BYTES_IN_FIFO
(USER_RAM_FIFO#(0:3)_NUM_BYTES_IN_FIFO@0)

Address: 47910

```
property user_ram_fifo1_allocate_num_bytes: int
```

Allocate memory for a FIFO buffer. Number of bytes should be sufficient to store users max transfer array size. Note that FLOAT32, INT32, and UINT32 require 4 bytes per value, and UINT16 require 2 bytes per value. Maximum size is limited by available memory. Care should be taken to conserve enough memory for other operations such as AIN_EF, Lua, Stream etc.

Register: USER_RAM_FIFO1_ALLOCATE_NUM_BYTES
(USER_RAM_FIFO#(0:3)_ALLOCATE_NUM_BYTES@1)

Address: 47902

```
property user_ram_fifo1_data_f32: float
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO1_DATA_F32 (USER_RAM_FIFO#(0:3)_DATA_F32@1)

Address: 47032

```
property user_ram_fifo1_data_i32: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO1_DATA_I32 (USER_RAM_FIFO#(0:3)_DATA_I32@1)

Address: 47022

```
property user_ram_fifo1_data_u16: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO1_DATA_U16 (USER_RAM_FIFO#(0:3)_DATA_U16@1)

Address: 47001

```
property user_ram_fifo1_data_u32: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO1_DATA_U32 (USER_RAM_FIFO#(0:3)_DATA_U32@1)

Address: 47012

```
property user_ram_fifo1_empty: int
```

Write any value to this register to efficiently empty, flush, or otherwise clear data from the FIFO.

Register: USER_RAM_FIFO1_EMPTY (USER_RAM_FIFO#(0:3)_EMPTY@1)

Address: 47932

```
property user_ram_fifo1_num_bytes_in_fifo: int
```

Poll this register to see when new data is available/ready. Each read of the FIFO buffer decreases this value, and each write to the FIFO buffer increases this value.

At any point in time, the following equation holds: Nbytes = Nwritten - Nread.

Register: USER_RAM_FIFO1_NUM_BYTES_IN_FIFO
(USER_RAM_FIFO#(0:3)_NUM_BYTES_IN_FIFO@1)

Address: 47912

```
property user_ram_fifo2_allocate_num_bytes: int
```

Allocate memory for a FIFO buffer. Number of bytes should be sufficient to store users max transfer array size. Note that FLOAT32, INT32, and UINT32 require 4 bytes per value, and UINT16 require 2 bytes per value. Maximum size is limited by available memory. Care should be taken to conserve enough memory for other operations such as AIN_EF, Lua, Stream etc.

Register: USER_RAM_FIFO2_ALLOCATE_NUM_BYTES
(USER_RAM_FIFO#(0:3)_ALLOCATE_NUM_BYTES@2)

Address: 47904

```
property user_ram_fifo2_data_f32: float
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO2_DATA_F32 (USER_RAM_FIFO#(0:3)_DATA_F32@2)

Address: 47034

```
property user_ram_fifo2_data_i32: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO2_DATA_I32 (USER_RAM_FIFO#(0:3)_DATA_I32@2)

Address: 47024

```
property user_ram_fifo2_data_u16: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO2_DATA_U16 (USER_RAM_FIFO#(0:3)_DATA_U16@2)

Address: 47002

```
property user_ram_fifo2_data_u32: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO2_DATA_U32 (USER_RAM_FIFO#(0:3)_DATA_U32@2)

Address: 47014

```
property user_ram_fifo2_empty: int
```

Write any value to this register to efficiently empty, flush, or otherwise clear data from the FIFO.

Register: USER_RAM_FIFO2_EMPTY (USER_RAM_FIFO#(0:3)_EMPTY@2)

Address: 47934

```
property user_ram_fifo2_num_bytes_in_fifo: int
```

Poll this register to see when new data is available/ready. Each read of the FIFO buffer decreases this value, and each write to the FIFO buffer increases this value.

At any point in time, the following equation holds: Nbytes = Nwritten - Nread.

Register: USER_RAM_FIFO2_NUM_BYTES_IN_FIFO
(USER_RAM_FIFO#(0:3)_NUM_BYTES_IN_FIFO@2)

Address: 47914

```
property user_ram_fifo3_allocate_num_bytes: int
```

Allocate memory for a FIFO buffer. Number of bytes should be sufficient to store users max transfer array size. Note that FLOAT32, INT32, and UINT32 require 4 bytes per value, and UINT16 require 2 bytes per value. Maximum size is limited by available memory. Care should be taken to conserve enough memory for other operations such as AIN_EF, Lua, Stream etc.

Register: USER_RAM_FIFO3_ALLOCATE_NUM_BYTES
(USER_RAM_FIFO#(0:3)_ALLOCATE_NUM_BYTES@3)

Address: 47906

```
property user_ram_fifo3_data_f32: float
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO3_DATA_F32 (USER_RAM_FIFO#(0:3)_DATA_F32@3)

Address: 47036

```
property user_ram_fifo3_data_i32: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO3_DATA_I32 (USER_RAM_FIFO#(0:3)_DATA_I32@3)

Address: 47026

```
property user_ram_fifo3_data_u16: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO3_DATA_U16 (USER_RAM_FIFO#(0:3)_DATA_U16@3)

Address: 47003

```
property user_ram_fifo3_data_u32: int
```

Generic FIFO buffer. Useful for passing ORDERED or SEQUENTIAL data between various endpoints, such as between a host and a Lua script. Use up to 4 FIFO buffers simultaneously->1 of each data type, all 4 different data types, or a mixture. e.g. FIFO0_DATA_U16 points to the same memory as other FIFO0 registers, such that there are a total of 4 memory blocks: FIFO0, FIFO1, FIFO2 and FIFO3. It is possible to write into a FIFO buffer using a different datatype than is being used to read out of it.

This register is a buffer. Underrun behavior - throws an error.

Register: USER_RAM_FIFO3_DATA_U32 (USER_RAM_FIFO#(0:3)_DATA_U32@3)

Address: 47016

```
property user_ram_fifo3_empty: int
```

Write any value to this register to efficiently empty, flush, or otherwise clear data from the FIFO.

Register: USER_RAM_FIFO3_EMPTY (USER_RAM_FIFO#(0:3)_EMPTY@3)

Address: 47936

```
property user_ram_fifo3_num_bytes_in_fifo: int
```

Poll this register to see when new data is available/ready. Each read of the FIFO buffer decreases this value, and each write to the FIFO buffer increases this value.

At any point in time, the following equation holds: Nbytes = Nwritten - Nread.

Register: USER_RAM_FIFO3_NUM_BYTES_IN_FIFO
(USER_RAM_FIFO#(0:3)_NUM_BYTES_IN_FIFO@3)

Address: 47916

```
property wait_us_blocking: int
```

Delays for x microseconds. Range is 0-100000. This operation is Blocking. While a blocking function is running no other registers can be read / written.

Register: WAIT_US_BLOCKING

Address: 61590

```
property watchdog_advanced_default: int
```

A single binary-encoded value where each bit is an advanced option. If bit 0 is set, IO_CONFIG_SET_CURRENT_TO_FACTORY will be done on timeout. If bit 1 is set, IO_CONFIG_SET_CURRENT_TO_DEFAULT will be done on timeout.

Register: WATCHDOG_ADVANCED_DEFAULT

Address: 61602

```
property watchdog_dac0_default: float
```

The voltage of DAC0 after a Watchdog timeout.

Register: WATCHDOG_DAC0_DEFAULT

Address: 61642

```
property watchdog_dac0_enable_default: int
```

Timeout action: Set to 1 to enable DAC0 update on watchdog timeout.

Register: WATCHDOG_DAC0_ENABLE_DEFAULT

Address: 61640

```
property watchdog_dac1_default: float
```

The voltage of DAC1 after a Watchdog timeout.

Register: WATCHDOG_DAC1_DEFAULT

Address: 61652

```
property watchdog_dac1_enable_default: int
```

Timeout action: Set to 1 to enable DAC1 update on watchdog timeout.

Register: WATCHDOG_DAC1_ENABLE_DEFAULT

Address: 61650

```
property watchdog_dio_direction_default: int
```

The direction input/output of the digital I/O after a Watchdog timeout. See DIO_DIRECTION.

Register: WATCHDOG_DIO_DIRECTION_DEFAULT

Address: 61634

```
property watchdog_dio_enable_default: int
```

Timeout action: Set to 1 to enable DIO update on watchdog timeout.

Register: WATCHDOG_DIO_ENABLE_DEFAULT

Address: 61630

```
property watchdog_dio_inhibit_default: int
```

This register can be used to prevent the watchdog from changing specific IO. See DIO_INHIBIT for more information.

Register: WATCHDOG_DIO_INHIBIT_DEFAULT

Address: 61636

```
property watchdog_dio_state_default: int
```

The state high/low of the digital I/O after a Watchdog timeout. See DIO_STATE.

Register: WATCHDOG_DIO_STATE_DEFAULT

Address: 61632

```
property watchdog_enable_default: int
```

Write a 1 to enable the watchdog or a 0 to disable. The watchdog must be disabled before writing any of the other watchdog registers (except for WATCHDOG_STRICT_CLEAR).

Register: WATCHDOG_ENABLE_DEFAULT

Address: 61600

```
property watchdog_reset_enable_default: int
```

Timeout action: Set to 1 to enable device-reset on watchdog timeout.

Register: WATCHDOG_RESET_ENABLE_DEFAULT

Address: 61620

```
property watchdog_startup_delay_s_default: int
```

This specifies the initial timeout period at device bootup. This is used until the first time the watchdog is cleared or timeout ... after that the normal timeout is used.

Register: WATCHDOG_STARTUP_DELAY_S_DEFAULT

Address: 61606

```
property watchdog_strict_clear: int
```

When running in strict mode, writing the key to this register is the only way to clear the watchdog. Writing to this register while not using strict mode will clear the watchdog.

Register: WATCHDOG_STRICT_CLEAR

Address: 61614

```
property watchdog_strict_enable_default: int
```

Set to 1 to enable strict mode.

Register: WATCHDOG_STRICT_ENABLE_DEFAULT

Address: 61610

```
property watchdog_strict_key_default: int
```

When set to strict mode, this is the value that must be written to the clear register.

Register: WATCHDOG_STRICT_KEY_DEFAULT

Address: 61612

```
property watchdog_timeout_s_default: int
```

When the device receives any communication over USB/Ethernet/WiFi, the watchdog timer is cleared. If the watchdog timer is not cleared within the timeout period, the enabled actions will be done.

Register: WATCHDOG_TIMEOUT_S_DEFAULT

Address: 61604

```
property wifi_apply_settings: int
```

Apply all new WiFi settings: IP, Subnet, Gateway, DHCP, SSID, Password. 1=Apply

Register: WIFI_APPLY_SETTINGS

Address: 49400

```
property wifi_dhcp_enable: int
```

Read the current Enabled/Disabled state of WiFi DHCP.

Register: WIFI_DHCP_ENABLE

Address: 49210

```
property wifi_dhcp_enable_default: int
```

The new Enabled/Disabled state of WiFi DHCP. Use WIFI_APPLY_SETTINGS.

Register: WIFI_DHCP_ENABLE_DEFAULT

Address: 49260

```
property wifi_firmware_update_status: int
```

CONFIGURING = 2920, IN_PROGRESS = 2921, REBOOTING = 2923, UPDATE_SUCCESS = 2924, UPDATE_FAILED = 2925.

Register: WIFI_FIRMWARE_UPDATE_STATUS

Address: 49454

```
property wifi_firmware_update_to_versionx: float
```

Start an update by using USB or Ethernet to write the desired version to update to.

Register: WIFI_FIRMWARE_UPDATE_TO_VERSIONX

Address: 49402

```
property wifi_gateway: str
```

Read the current gateway of WiFi.

Register: WIFI_GATEWAY

Address: 49204

```
property wifi_gateway_default: str
```

The new gateway of WiFi. Use WIFI_APPLY_SETTINGS.

Register: WIFI_GATEWAY_DEFAULT

Address: 49254

```
property wifi_ip: str
```

Read the current IP address of WiFi.

Register: WIFI_IP

Address: 49200

```
property wifi_ip_default: str
```

The new IP address of WiFi. Use WIFI_APPLY_SETTINGS.

Register: WIFI_IP_DEFAULT

Address: 49250

```
property wifi_mac: str
```

The MAC address of the WiFi module.

Register: WIFI_MAC

Address: 60024

```
property wifi_password_default: str
```

Write the password for the WiFi network, then use WIFI_APPLY_SETTINGS.

Register: WIFI_PASSWORD_DEFAULT

Address: 49350

```
property wifi_rssi: float
```

WiFi RSSI (signal strength). Typical values are -40 for very good, and -75 for very weak. The T7 microcontroller only gets a new RSSI value from the WiFi module when WiFi communication occurs.

Register: WIFI_RSSI

Address: 49452

```
property wifi_ssid: str
```

Read the current SSID (network name) of WiFi.

Register: WIFI_SSID

Address: 49300

```
property wifi_ssid_default: str
```

The new SSID (network name) of WiFi. Use WIFI_APPLY_SETTINGS.

Register: WIFI_SSID_DEFAULT

Address: 49325

```
property wifi_status: int
```

Status Codes: ASSOCIATED = 2900, ASSOCIATING = 2901, ASSOCIATION_FAILED = 2902, UNPOWERED = 2903, BOOTING = 2904, START_FAILED = 2905, APPLYING_SETTINGS = 2906, DHCP_STARTED = 2907, OTHER = 2909

Register: WIFI_STATUS

Address: 49450

```
property wifi_subnet: str
```

Read the current subnet of WiFi.

Register: WIFI_SUBNET

Address: 49202

```
property wifi_subnet_default: str
```

The new subnet of WiFi. Use WIFI_APPLY_SETTINGS.

Register: WIFI_SUBNET_DEFAULT

Address: 49252

```
property wifi_udp_discovery_only: int
```

Restricts which Modbus operations are allowed over UDP. When set to 1, only the registers needed for discovering units can be read and any other operation will throw an error.

Register: WIFI_UDP_DISCOVERY_ONLY

Address: 49215

```
property wifi_udp_discovery_only_default: int
```

The Enabled/Disabled state of WIFI_UDP_DISCOVERY_ONLY after a power-cycle to the device.

Register: WIFI_UDP_DISCOVERY_ONLY_DEFAULT

Address: 49265

```
property wifi_version: float
```

The current firmware version of the WiFi module, if available.

Register: WIFI_VERSION

Address: 60008

IO — HILSTER's proprietary IO-hardware

Introduction

HILSTER offers prorietary IO-hardware that is DIN rail mountable. Currently there are four different types and sizes available.

You need the `io` feature in your license to unlock the IO hardware features.

For every device an abstraction is available. To read inputs or write outputs you simply have to assign a value to an output or read an input. The abstractors offer members with the name of the inputs and outputs, e.g. `io.R6` for relay 6, `io.DO0` for digital output 0 or `io.AI3` for analog input 3.

For example to switch on a relay `io.R6` you have to assign `True` to it.

```
io.R6 = True
```

To reset the reley you have to assign `False`.

```
io.R6 = False
```

To set the digital output pin `io.D3` you have to assign `True` to it.

```
io.D3 = True
```

To reset the digital output pin assign `False` to it.

```
io.R6 = False
```

To set an analog (pwm) level to 50 % you have to assign an integer value knowing the underlying pwm type. In the following example an 8-bit-pwm is assumed.

```
io.AO0 = 0xf7  # 50 % pwm level
```

Finally to read an analog level you have to read you simply have to read an analog input

```
analog_value = io.AI3
```

IO Mini



Features:

- 6 relays (230 V/6 A)
- 8 analog/digital inputs
- 4 digital outputs (2 A)
- 1 UART
- 1 SPI
- 1 I2C
- RTC

A sample implementation and usage can be found here:

```
from htf.io import IOMini as IO
io = IO("/dev/ttyACM0")
# set D1
io.D1 = True
# reset D1
io.D1 = False
#read A0
analog_value = io.A0
#read D0
digital_value = io.D0
# etc.
```

```
class htf.io.IOMini(port: str, timeout: int | float = 1.0)
```

Initialize an IOMini.

Parameters:

- **port** – the name of the serial connection
- **timeout=1.0** – the time-out in seconds

```
A0
```

Analog input A0

```
A1
```

Analog input A1

```
A2
```

Analog input A2

```
A3
```

Analog input A3

```
A4
```

Analog input A4

```
A5
```

Analog input A5

```
D0
```

Digital output D0

```
D1
```

Digital output D1

```
D2
```

Digital output D2

```
D3
```

Digital output D3

```
D4
```

Digital output D4

```
D5
```

Digital output D5

```
D6
```

Digital output D6

```
D7
```

Digital output D7

```
IN0
```

Digital input IN0

```
IN1
```

Digital input IN1

```
__del__() → None
```

Automatically close serial connection on deletion.

```
close() → None
```

Close the serial connection.

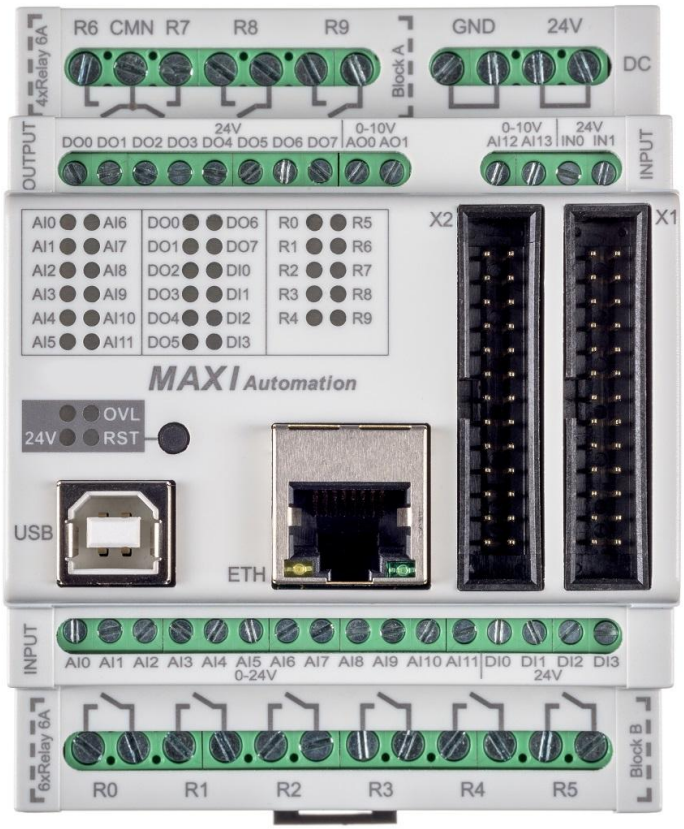
```
get_board_name() → str
```

Return the board name read from the device.

```
get_version() → int
```

Return the version read from the device.

IO Maxi



Features:

- 10 relays (230V/6A)
- 2 analog outputs (0 to 10 V)
- 12 analog inputs (24 V)
- 12 digital inputs (24 V)
- 8 digital outputs (24 V)
- 2 UART
- 1 SPI
- 1 I2C
- RTC
- Ethernet

A sample implementation and usage can be found here:

```
from htf.io import IOMaxi as IO
io = IO("/dev/ttyACM0")
# set R6
io.R6 = True
# reset R6
io.R6 = False
#read AI0
analog_value = io.AI0
```

```
#read DI0
digital_value = io.DI0
# etc.
```

```
class htf.io.IOMaxi(port: str, timeout: int | float = 1.0)
```

Initialize an IOMaxi.

Parameters:

- **port** – the name of the serial connection
- **timeout=1.0** – the time-out in seconds

```
A0
```

Analog input A0

```
A1
```

Analog input A1

```
A2
```

Analog input A2

```
A3
```

Analog input A3

```
A4
```

Analog input A4

```
A5
```

Analog input A5

```
A6
```

Analog input A6

```
A7
```

Analog input A7

```
A8
```

Analog input A8

```
A9
```

Analog input A9

```
D0
```

Digital output D0

```
D1
```

Digital output D1

```
D10
```

Digital output D10

```
D11
```

Digital output D11

```
D2
```

Digital output D2

```
D3
```

Digital output D3

```
D4
```

Digital output D4

```
D5
```

Digital output D5

```
D6
```

Digital output D6

```
D7
```

Digital output D7

```
D8
```

Digital output D8

```
D9
```

Digital output D9

```
IN0
```

Digital input IN0

```
IN1
```

Digital input IN1

```
R0
```

Relay R0

```
R1
```

Relay R1

```
R2
```

Relay R2

```
R3
```

Relay R3

```
R4
```

Relay R4

```
R5
```

Relay R5

```
R6
```

Relay R6

```
R7
```

Relay R7

```
R8
```

Relay R8

```
R9
```

Relay R9

```
__del__() → None
```

Automatically close serial connection on deletion.

```
close() → None
```

Close the serial connection.

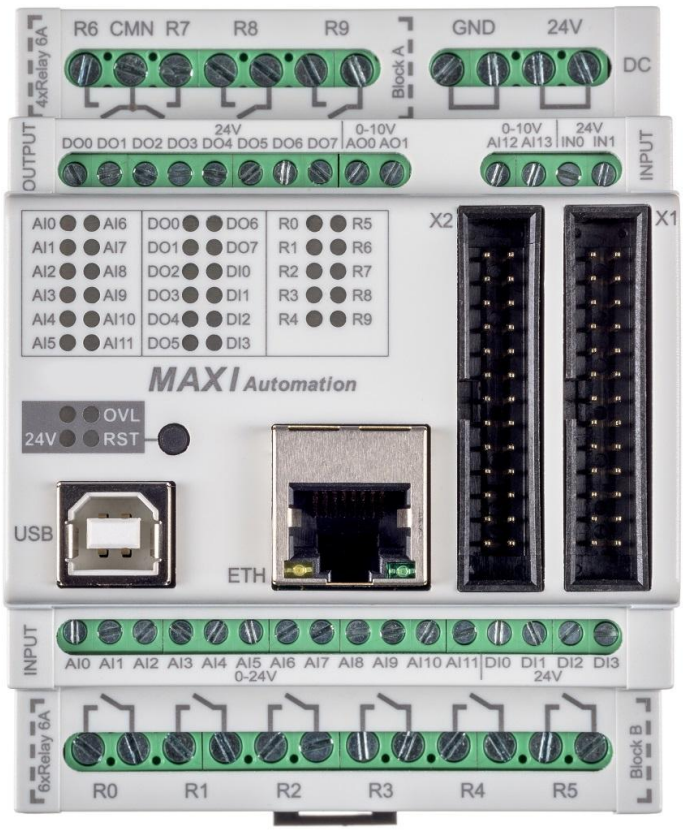
```
get_board_name() → str
```

Return the board name read from the device.

```
get_version() → int
```

Return the version read from the device.

IO Maxi Automation



Features:

- 10 relays (230 V/6 A)
- 2 analog outputs (0 to 24 V)
- 2 analog inputs (10 V)
- 12 analog/digital inputs (24 V)
- 4 digital inputs (24 V)
- 8 digital outputs (24 V)
- 2 UART
- 1 SPI

- 1 I2C
- RTC
- Ethernet

A sample impelentation and usage can be found here:

```
from htf.io import IOMaxiAutomation as IO
io = IO("/dev/ttyACM0")
# set R6
io.R6 = True
# reset R6
io.R6 = False
#read AI0
analog_value = io.AI0
#read DI0
digital_value = io.DI0
# etc.
```

```
class htf.io.IOMaxiAutomation(port: str, timeout: int | float = 1.0)
```

Initialize an IOMaxiAutomation.

Parameters:

- **port** – the name of the serial connection
- **timeout=1.0** – the time-out in seconds

```
AI0
```

Analog input AI0

```
AI1
```

Analog input AI1

```
AI10
```

Analog input AI10

```
AI11
```

Analog input AI11

```
AI12
```

Analog input AI12

```
AI13
```

Analog input AI13

```
AI2
```

Analog input AI2

```
AI3
```

Analog input AI3

```
AI4
```

Analog input AI4

```
AI5
```

Analog input AI5

```
AI6
```

Analog input AI6

```
AI7
```

Analog input AI7

```
AI8
```

Analog input AI8

```
AI9
```

Analog input AI9

```
AO0
```

Analog output AO0

```
AO1
```

Analog output AO1

```
DI0
```

Digital input DI0

```
DI1
```

Digital input DI1

```
DI2
```

Digital input DI2

```
DI3
```

Digital input DI3

```
DO0
```

Digital output DO0

```
DO1
```

Digital output DO1

```
DO2
```

Digital output DO2

`DO3`

Digital output DO3

`DO4`

Digital output DO4

`DO5`

Digital output DO5

`DO6`

Digital output DO6

`DO7`

Digital output DO7

`IN1`

Digital input IN1

`IN2`

Digital input IN2

`R0`

Relay R0

`R1`

Relay R1

`R2`

Relay R2

```
R3
```

Relay R3

```
R4
```

Relay R4

```
R5
```

Relay R5

```
R6
```

Relay R6

```
R7
```

Relay R7

```
R8
```

Relay R8

```
R9
```

Relay R9

```
__del__() → None
```

Automatically close serial connection on deletion.

```
close() → None
```

Close the serial connection.

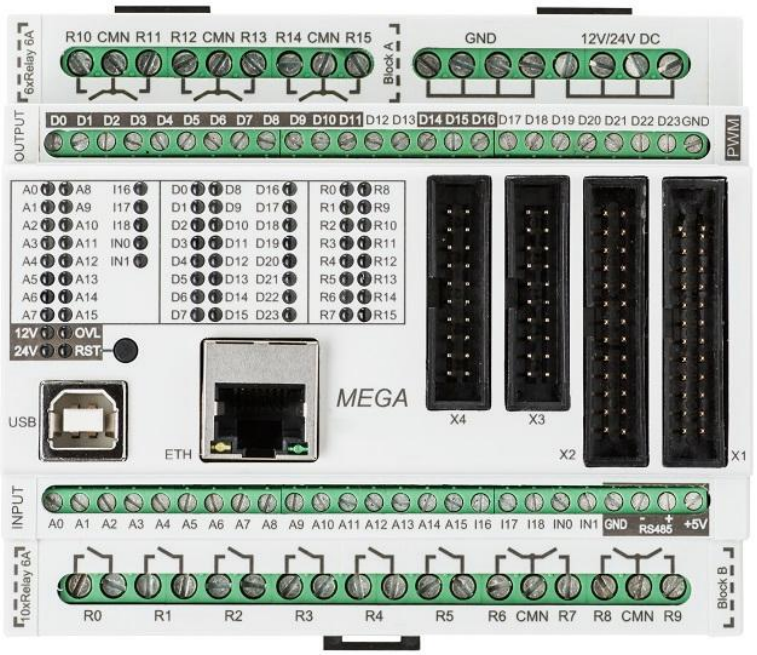
```
get_board_name() → str
```

Return the board name read from the device.

```
get_version() → int
```

Return the version read from the device.

IO Mega



Features:

- 16 relays (230 V/6 A)
- 2 analog outputs (0 to 10 V)
- 21 analog inputs (24 V)
- 12 digital outputs (24 V)
- 2 UART
- 1 RS485
- 1 SPI
- 1 I2C
- RTC
- Ethernet

A sample implementation and usage can be found here:

```
from htf.io import IOMega as IO
io = IO("/dev/ttyACM0")
# set R6
io.R6 = True
# reset R6
io.R6 = False
#read AI0
analog_value = io.AI0
#read DI0
digital_value = io.DI0
# etc.
```

```
class htf.io.IOMega(port: str, timeout: int | float = 1.0)
```

Initialize an IOMega.

Parameters:

- **port** – the name of the serial connection
- **timeout=1.0** – the time-out in seconds

```
A0
```

Analog input A0

```
A1
```

Analog input A1

```
A10
```

Analog input A10

```
A11
```

Analog input A11

```
A12
```

Analog input A12

```
A13
```

Analog input A13

```
A14
```

Analog input A14

```
A15
```

Analog input A15

```
A2
```

Analog input A2

```
A3
```

Analog input A3

```
A4
```

Analog input A4

```
A5
```

Analog input A5

```
A6
```

Analog input A6

```
A7
```

Analog input A7

```
A8
```

Analog input A8

```
A9
```

Analog input A9

```
D0
```

Digital output D0

```
D1
```

Digital output D1

```
D10
```

Digital output D10

```
D11
```

Digital output D11

```
D12
```

Digital output D12

```
D13
```

Digital output D13

```
D14
```

Digital output D14

```
D15
```

Digital output D15

`D16`

Digital output D16

`D17`

Digital output D17

`D18`

Digital output D18

`D19`

Digital output D19

`D2`

Digital output D2

`D3`

Digital output D3

`D4`

Digital output D4

`D5`

Digital output D5

`D6`

Digital output D6

`D7`

Digital output D7

```
D8
```

Digital output D8

```
D9
```

Digital output D9

```
I16
```

Digital input I16

```
I17
```

Digital input I17

```
I18
```

Digital input I18

```
IN0
```

Digital input IN0

```
IN1
```

Digital input IN1

```
R0
```

Relay R0

```
R1
```

Relay R1

```
R10
```

Relay R10

```
R11
```

Relay R11

```
R12
```

Relay R12

```
R13
```

Relay R13

```
R14
```

Relay R14

```
R15
```

Relay R15

```
R2
```

Relay R2

```
R3
```

Relay R3

```
R4
```

Relay R4

```
R5
```

Relay R5

```
R6
```

Relay R6

```
R7
```

Relay R7

```
R8
```

Relay R8

```
R9
```

Relay R9

```
__del__() → None
```

Automatically close serial connection on deletion.

```
close() → None
```

Close the serial connection.

```
get_board_name() → str
```

Return the board name read from the device.

```
get_version() → int
```

Return the version read from the device.

Changelog

htf-io-4.3.0

- ship `cp312-abi3` wheels that are compatible with Python 3.12 and newer versions

htf-io-4.2.1

- implement Cyber Resilience Act Conformity (CRA)
- update dependencies

htf-io-4.2.0

- add support for Xentara

htf-io-4.1.5

- add support for Python 3.13

htf-io-4.1.4

- add requirements for validation purposes

htf-io-4.1.3

- add build environment data for validation purposes

htf-io-4.1.2

- `htf-io` can now be used standalone

htf-io-4.1.1

- add support for Python 3.12

htf-io-4.1.0

- add support for LabJack DAQ devices

htf-io-4.0.0

- remove all camel-case methods
- add type-hints
- use `hlm-3.1`
- use `oser-3.1`

htf-io-3.0.2

- add support for Python 3.11

htf-io-3.0.1

- fix an import error

htf-io-3.0.0

- extract `htf-io` from `htf`

Arduino — Simple IO using open source hardware

Generic Arduino Support

Generic Arduino abstractors support digital input with and without internal pullup, digital output, analog input and analog output (pwm).

Pins are read and written by simply reading from or writing to member variables of the used abstractor.

Digital output pins are accessed by writing a boolean or integer value to `do<N>` or `digital_output<N>` where `<N>` is the pin number. Additionally you can call the method `digital_output(pin)` if you like to. When writing a digital output pin the pin is automatically set to digital output mode.

Digital input pins can be accessed by reading `di<N>` or `digital_input<N>` where `<N>` is the pin number. Additionally you can call the method `digital_input(pin)` if you like to. When reading a digital input pin the pin is automatically set to digital input mode.

Digital input pins with internal pullup can be accessed by reading `dip<N>` or `digital_input_pullup<N>` where `<N>` is the pin number. Additionally you can call the method `digital_input_pullup(pin)` if you like to. When reading a digital input pin with internal pullup the pin is automatically set to digital input mode with pullup enabled.

Analog output or pwm pins are accessed by writing an integer value (0 .. 255) to `pwm<N>`, `ao<N>` or `analog_output<N>` where `<N>` is the pin number. Additionally you can call the method `analog_output(pin)` if you like to. When writing an analog output (pwm) pin the pin is automatically set to analog output (pwm) mode. There are different names for the same pins. This is useful to give you code a better understanding of what happens when the pin is set. A pwm pin with a low-pass filter connected results in an analog voltage so the user would rather use `ao<N>` instead of `pwm<N>`.

Analog input pins are accessed by reading `ai<N>` or `analog_input<N>` where `<N>` is the pin number. Additionally you can call the method `analog_input(pin)` or `pwm(pin)` if you like to. When reading an analog input the pin is automatically set to analog input mode.

You need the `io` feature in your license to unlock the Arduino features.

Arduino Nano

`htf.arduino.ArduinoNano` supports 12 digital input pins with and without internal pullup (D2 .. D13), 12 digital output pins (D2 .. D13), 8 analog input pins (A0 .. A7) and 6 analog output (pwm) pins (D3, D5, D6, D9, D10, and D11).

To setup the Arduino Nano use:

```
from htf.arduino import ArduinoNano
a = ArduinoNano("/dev/ttyUSB0")
# set pin D1 to 1
a.do3 = 1
```

```
# set pin D1 to 0
a.do3 = 0
# read pin D4
a.di4
>>> 0
# read pin D4 with internal pullup
a.dip4
>>> 1
# read A3
a.ai3
>>> 486
# set pwm level to 50 % on D3
a.pwm3 = 0x8f
# set analog voltage to 5 V on D5
a.ao5 = 0xff
```

```
class htf.arduino.ArduinoNano(port: str, timeout: int | float = 0.5)
```

Initialize an ArduinoNano.

Parameters:

- **port** – the name of the serial connection
- **timeout=0.5** – the time-out in seconds

```
__del__() → None
```

Automatically close serial connection on deletion.

```
analog_input(pin: int) → int
```

Read an analog input pin.

Parameters:
pin – the pin to be read
Returns:
the analog value of `pin`
Return type:
int

```
analog_output(pin: int, value: int) → None
```

Set `pin` to analog level `value`. This is useful if `pin` is connected to a low-pass filter.

Parameters:

- **pin** – the pin to be used
- **value** – the analog output level (0 .. 255)

```
close() → None
```

Close the serial connection.

```
digital_input(pin: int) → int
```

Read a digital input pin.

Parameters:
pin – the pin to be read
Returns:
the pin state of `pin`
Return type:
int

```
digital_input_pullup(pin: int) → int
```

Read a digital input pin with internal pullup enabled.

Parameters:
pin – the pin to be read
Returns:
the pin state of `pin`
Return type:
int

```
digital_output(pin: int, state: int) → None
```

Write a digital output pin.

Parameters:

- **pin** – the pin to be written
- **state** – the state for `pin`

```
get_board_name() → str
```

Return the board name read from the device.

```
get_version() → int
```

Return the version read from the device.

```
pwm(pin: int, value: int) → None
```

Set `pin` to pwm-level `value`.

Parameters:

- **pin** – the pin to be used
- **value** – the pwm level (0 .. 255)

Arduino Leonardo

`htf.arduino.ArduinoLeonardo` supports 14 digital input pins with and without internal pullup (D0 .. D13), 14 digital output pins (D0 .. D13), 12 analog input pins (A0 .. A5, A6 .. A11 on digital pins D4, D6, D8, D9, D10, and D12) and 6 analog output (pwm) pins (D3, D5, D6, D9, D10, D11 and D13).

To setup the Arduino Leonardo use:

```
from htf.arduino import ArduinoLeonardo
a = ArduinoLeonardo("/dev/ttyACM0")
# set pin D1 to 1
a.do3 = 1
# set pin D1 to 0
a.do3 = 0
# read pin D4
a.di4
>>> 0
# read pin D4 with internal pullup
a.dip4
>>> 1
# read A3
a.ai3
>>> 486
# set pwm level to 50 % on D3
a.pwm3 = 0x7f
# set analog voltage to 5 V on D5
a.ao5 = 0xff
```

```
class htf.arduino.ArduinoLeonardo(port: str, timeout: int | float = 0.5)
```

Initialize an ArduinoLeonardo.

Parameters:

- **port** – the name of the serial connection
- **timeout=0.5** – the time-out in seconds

```
__del__() → None
```

Automatically close serial connection on deletion.

```
analog_input(pin: int) → int
```

Read an analog input pin.

Parameters:
pin – the pin to be read
Returns:
the analog value of `pin`
Return type:
int

```
analog_output(pin: int, value: int) → None
```

Set `pin` to analog level `value`. This is useful if `pin` is connected to a low-pass filter.

Parameters:

- **pin** – the pin to be used
- **value** – the analog output level (0 .. 255)

```
close() → None
```

Close the serial connection.

```
digital_input(pin: int) → int
```

Read a digital input pin.

Parameters:
pin – the pin to be read
Returns:
the pin state of `pin`
Return type:
int

```
digital_input_pullup(pin: int) → int
```

Read a digital input pin with internal pullup enabled.

Parameters:
pin – the pin to be read
Returns:
the pin state of `pin`
Return type:
int

```
digital_output(pin: int, state: int) → None
```

Write a digital output pin.

Parameters:

- **pin** – the pin to be written
- **state** – the state for `pin`

```
get_board_name() → str
```

Return the board name read from the device.

```
get_version() → int
```

Return the version read from the device.

```
pwm(pin: int, value: int) → None
```

Set `pin` to pwm-level `value`.

Parameters:

- **pin** – the pin to be used
- **value** – the pwm level (0 .. 255)

Arduino Leonardo Digital Input

`htf.arduino.ArduinoLeonardoDigitalInput` can be used to read all 14 digital inputs (pins 0 .. 13) and to count edges (using pin-change interrupts) on pins 0, 1, 2 and 3.

To use the Arduino Leonardo Digital Inputs:

```
di = ArduinoLeonardoDigitalInput("COM4")
print("Version:", di.get_version())
data = di.get()
print(data) # a dictionary containing all data
```

All entries in `data` starting with `"d"` are digital input states, ie. `0` or `1`. All entries in `data` starting with `"c"` are edge counts, ie. `unsigned long int`. Edge counts can be reset using `reset`.

```
class htf.arduino.ArduinoLeonardoDigitalInput(port: str, reset_on_init: bool = True,
timeout: int | float = 0.5)
```

`ArduinoLeonardoDigitalInput` reads inputs (on pins 0 .. 13) and counts edges (on pins 0, 1, 2 and 3) using pin change interrupts on Arduino Leonardo.

Serial connection: 115200 baud, 8N1.

Parameters:

- **port** – the com port the Arduino Leonardo listens to running the digital input application.
- **reset_on_init=True** – reset internal edge counters on initialization.
- **timeout=0.5** – the timeout for serial communication in seconds.

```
close() → None
```

Close serial connection.

```
get() → Dict[str, int]
```

Get current data, ie. digital input states and edge counts.

Returns:
the current data containing digital port states and counter values.

E.g. {'d0':1,'d1':1,'d2':0,'d3':1,'d4':1,'d5':1,'d6':1,'d7':1,'d8':1,'d9':1,'d10':1,'d11':1,'d12':1, 'd13':1,'c0':0,'c1':0,'c2':77,'c3':102}

Return type:
dict

```
get_version() → bytes
```

Get software version.

Returns:
the software version.
Return type:
bytes

```
reset() → None
```

Reset all edge counters.

DAQmx — National Instruments DAQmx C-api wrapper

htf has built-in support for National Instruments DAQmx.

Adapters

DAQmx is a python wrapper to the National Instruments DAQmx driver.

Adapters are abstract helper classes to simply use the National Instruments hardware.

You need the `io` feature in your license to unlock the DAQmx features.

Digital Input Single Line

Simple usage:

```
from htf.daqmx import DigitalInputSingleLine

di = DigitalInputSingleLine("cDaq1Mod1/line0")
print(di.get())
```

Usage in delegation context:

```
from htf.daqmx import DigitalInputSingleLine

class HardwareAbstraction(DelegatorMixin):
    def __init__(self):
        self.di = DigitalInputSingleLine("cDaq1Mod1/line0")

ha = HardwareAbstraction()
# get current value
print(ha.di)
```

```
class htf.daqmx.DigitalInputSingleLine(descriptor: str)
```

`DigitalInputSingleLine` is used to access a single digital input line on a DAQmx device.

A DAQmx task is automatically created and deleted in the constructor and destructor.

Parameters:
descriptor – The DAQmx device and line descriptor, eg. *cDaq1Mod1/line0*.

```
close() → None
```

Close the abstractor by clearing the task.

```
get() → bool
```

Return the current pin state.

```
set(state: bool = True) → None
```

The setter for a delegate.

Parameters:
value – the value to be set.

Note

This method is abstract.

Digital Input Port

Simple usage:

```
from htf.daqmx import DigitalInputPort

di = DigitalInputPort("cDaq1Mod1/port0")
print(di.get())
```

Usage in delegation context:

```
from htf.daqmx import DigitalInputPort

class HardwareAbstraction(DelegatorMixin):
   def __init__(self):
      self.di = DigitalInputPort("cDaq1Mod1/port0")

ha = HardwareAbstraction()
# get current value
print(ha.di)
```

```
class htf.daqmx.DigitalInputPort(descriptor: str)
```

`DigitalInputPort` is used to access a digital input port on a DAQmx device.

A DAQmx task is automatically created and deleted in the constructor and destructor.

Parameters:
descriptor (*str*) – The DAQmx device and line descriptor, eg. *cDaq1Mod1/port0*.

```
close() → None
```

Close the abstractor by clearing the task.

```
get() → int
```

Return the current port state.

Digital Output Single Line

Simple usage:

```
from htf.daqmx import DigitalOutputSingleLine

do = DigitalOutputSingleLine("cDaq1Mod1/line0")
print(do.set(1))
```

Usage in delegation context:

```
from htf.daqmx import DigitalOutputSingleLine

class HardwareAbstraction(DelegatorMixin):
    def __init__(self):
        self.do = DigitalOutputSingleLine("cDaq1Mod1/line0")

ha = HardwareAbstraction()
# set current value to 1
ha.do = 1
```

```
class htf.daqmx.DigitalOutputSingleLine(descriptor: str)
```

`DigitalOutputSingleLine` is used to access a single digital output line on a DAQmx device.

A DAQmx task is automatically created and deleted in the constructor and destructor.

Parameters:
descriptor – The DAQmx device and line descriptor, eg. *cDaq1Mod1/line0*.

```
close() → None
```

Close the abstractor by clearing the task.

```
get() → bool
```

Read the current pin state.

```
reset() → None
```

Reset the pin state.

```
set(state: bool = True) → None
```

Set the pin state to `state`.

Parameters:
state=True – new state for output pin.

Digital Output Port

Simple usage:

```
from htf.daqmx import DigitalOutputPort

do = DigitalOutputPort("cDaq1Mod1/port0")
do.set(0x12345678)
```

Usage in delegation context:

```
from htf.daqmx import DigitalOutputPort

class HardwareAbstraction(DelegatorMixin):
    def __init__(self):
        self.do = DigitalOutputPort("cDaq1Mod1/port0")

ha = HardwareAbstraction()
# set current value to 0x12345678
ha.do = 0x12345678
```

```
class htf.daqmx.DigitalOutputPort(descriptor: str)
```

`DigitalOutputPort` is used to access a digital output port on a DAQmx device.

A DAQmx task is automatically created and deleted in the constructor and destructor.

Parameters:
descriptor (*str*) – The DAQmx device and line descriptor, eg. *cDaq1Mod1/port0*.

```
close() → None
```

Close the abstractor by clearing the task.

```
get() → int
```

Return the current port state.

```
set(value: int) → None
```

Set the port to `value`.

Parameters:
value (*float*) – new value for output port

Analog Voltage Input Single Line

Simple usage:

```
from htf.daqmx import AnalogVoltageInputSingleLine

ai = AnalogVoltageInputSingleLine("cDaq1Mod1/line0")
print(ai.get())
```

Usage in delegation context:

```
from htf.daqmx import AnalogVoltageInputSingleLine

class HardwareAbstraction(DelegatorMixin):
    def __init__(self):
        self.ai = AnalogVoltageInputSingleLine("cDaq1Mod1/line0")

ha = HardwareAbstraction()
# get current value
print(ha.ai)
```

```
class htf.daqmx.AnalogVoltageInputSingleLine(descriptor: str, minimum_voltage: float =
-10.0, maximum_voltage: float = 10.0, terminal_config: int = -1)
```

`AnalogVoltageInputSingleLine` is used to access a single analog output line on a DAQmx device.

A DAQmx task is automatically created and deleted in the constructor and destructor.

Parameters:

- **descriptor** – The DAQmx device and line descriptor, eg. *cDaq1Mod1/line0*
- **minimum_voltage=-10.0** – the minimum voltage.
- **maximum_voltage=10.0** – the maximum voltage.

- **terminal_config=DAQmx_Val_Cfg_Default** – the terminal configuration for DAQmxCreateAIVoltageChan(). Possible values: DAQmx_Val_RSE, DAQmx_Val_NRSE, DAQmx_Val_Diff, DAQmx_Val_PseudoDiff.

```
close() → None
```

Close the abstractor by clearing the task.

```
get() → float
```

Return the current pin value.

```
set(state: bool = True) → None
```

The setter for a delegate.

Parameters:
value – the value to be set.

> **Note**
> This method is abstract.

Analog Voltage Input Single Line Accurate (averaging analog input)

Simple usage:

```
from htf.daqmx import AnalogVoltageInputSingleLineAccurate

ai = AnalogVoltageInputSingleLineAccurate("cDaq1Mod3/ai0", numberOfSamples=100, timeout=3.0,
sampleFrequency=100)
print(ai.get())
```

Usage in delegation context:

```
from htf.daqmx import AnalogVoltageInputSingleLineAccurate

class HardwareAbstraction(DelegatorMixin):
    def __init__(self):
        self.ai = AnalogVoltageInputSingleLineAccurate("cDaq1Mod3/ai0", numberOfSamples=100,
timeout=3.0, sampleFrequency=100)

ha = HardwareAbstraction()
# get current value
print(ha.ai)
```

```
class htf.daqmx.AnalogVoltageInputSingleLineAccurate(descriptor: str, minimum_voltage:
float = -10.0, maximum_voltage: float = 10.0, number_of_samples: int = 2, sample_frequency:
float = 1000.0, timeout: float = 1.0, terminal_config: int = -1)
```

`AnalogVoltageInputSingleLineAccurate` is used to access a single analog input line on a DAQmx device. It acquires some samples and builds the average value of them.

A DAQmx task is automatically created and deleted in the constructor and destructor.

Parameters:

- **descriptor** – The DAQmx device and line descriptor, eg. *cDaq1Mod1/line0*
- **minimum_voltage=-10.0** – the minimum voltage.
- **maximum_voltage=10.0** – the maximum voltage.
- **number_of_samples=2** – the number of samples to acquire for one average value.
- **sample_frequency=1000.0** – the sample frequency in Hertz.
- **timeout=1.0** – the timeout when acquiring a new value in seconds.
- **terminal_config=DAQmx_Val_Cfg_Default** – the terminal configuration for DAQmxCreateAIVoltageChan(). Possible values: DAQmx_Val_RSE, DAQmx_Val_NRSE, DAQmx_Val_Diff, DAQmx_Val_PseudoDiff.

```
close() → None
```

Close the abstractor by clearing the task.

```
get() → float
```

Return the current pin value.

```
set(value: bool = True) → None
```

The setter for a delegate.

Parameters:
value – the value to be set.

Note

This method is abstract.

Analog Voltage Input Port

Simple usage:

```
from htf.daqmx import AnalogVoltageInputPort
```

```
aport = AnalogVoltageInputPort("cDaq1Mod3/ai0:2", numberOfLines=3, numberOfSamples=2, timeout=3.0,
sampleFrequency=1000)
print(aport.get())
# [
#     [line0_sample0, line0_sample1],
#     [line1_sample0, line1_sample1],
#     [line2_sample0, line2_sample1]
# ]
```

Usage in delegation context:

```
from htf.daqmx import AnalogVoltageInputPort

class HardwareAbstraction(DelegatorMixin):
    def __init__(self):
        self.aport = AnalogVoltageInputPort("cDaq1Mod3/ai0:2", numberOfLines=3, numberOfSamples=2,
timeout=3.0, sampleFrequency=1000)


ha = HardwareAbstraction()
# get current values
print(ha.aport)
# [
#     [line0_sample0, line0_sample1],
#     [line1_sample0, line1_sample1],
#     [line2_sample0, line2_sample1]
# ]
```

```
class htf.daqmx.AnalogVoltageInputPort(descriptor: str, minimum_voltage: float = -10.0,
maximum_voltage: float = 10.0, number_of_lines: int = 32, number_of_samples: int = 2,
sample_frequency: float = 1000.0, timeout: float = 1.0, terminal_config: int = -1,
continuous_mode: bool = False)
```

`AnalogVoltageInputPort` is used to access a single analog input port on a DAQmx device.

A DAQmx task is automatically created and deleted in the constructor and destructor.

Parameters:

- **descriptor** – The DAQmx device and line descriptor, eg. `cDaq1Mod1/line0`.
- **minimum_voltage=-10.0** – the minimum voltage.
- **maximum_voltage=10.0** – the maximum voltage.
- **number_of_lines=32** – the number of lines defined in the descriptor. For example the descriptor "cDaq1Mod3/ai0:31" defines 32 lines.
- **number_of_samples=2** – the number of samples to acquire for one average value.
- **sample_frequency=1000.0** – the sample frequency in Hertz.
- **timeout=1.0** – the timeout when acquiring a new value in seconds.
- **terminal_config=DAQmx_Val_Cfg_Default** – the terminal configuration for DAQmxCreateAIVoltageChan(). Possible values: DAQmx_Val_RSE, DAQmx_Val_NRSE, DAQmx_Val_Diff, DAQmx_Val_PseudoDiff.

- **continuous_mode=False** – if set to `True` continuous sampling is done and get() becomes blocking. Simply read samples in a loop.

```
close() → None
```

Close the abstractor by clearing the task.

```
get() → List[List[float]]
```

Return the current analog value.

Returned data layout:

```
[
    [line0 sample0, line0 sample1, ..., line0 sampleN-1],
    [line1 sample0, line1 sample1, ..., line1 sampleN-1],
    ...
    [lineM-1 sample0, lineM-1 sample1, ..., lineM-1 sampleN-1]
]
```

```
set(state: bool = True) → None
```

The setter for a delegate.

Parameters:
value – the value to be set.

Note

This method is abstract.

Analog Voltage Output Single Line

Simple usage:

```
from htf.daqmx import AnalogVoltageOutputSingleLine

ao = AnalogVoltageOutputSingleLine("cDaq1Mod1/line0")
print(ao.set(3.0))
```

Usage in delegation context:

```
from htf.daqmx import AnalogVoltageOutputSingleLine

class HardwareAbstraction(DelegatorMixin):
```

```
    def __init__(self):
        self.ao = AnalogVoltageOutputSingleLine("cDaq1Mod1/line0")

ha = HardwareAbstraction()
# set value
ha.ao = 1.337
```

```
class htf.daqmx.AnalogVoltageOutputSingleLine(descriptor: str, minimum_voltage: float =
-10.0, maximum_voltage: float = 10.0)
```

`AnalogVoltageOutputSingleLine` is used to access a single analog output line on a DAQmx device.

A DAQmx task is automatically created and deleted in the constructor and destructor.

Parameters:

- **descriptor** – The DAQmx device and line descriptor, eg. *cDaq1Mod1/line0*
- **minimum_voltage=-10.0** – the minimum voltage.
- **maximum_voltage=10.0** – the maximum voltage.

```
close() → None
```

Close the abstractor by clearing the task.

```
get() → float
```

Return the last written analog value.

```
set(value: float) → None
```

Set the pin value to `value`.

Parameters:
value – new state for output pin.

Analog Voltage Output Port

Simple usage:

```
from htf.daqmx import AnalogVoltageOutputPort

aport = AnalogVoltageOutputPort("cDaq1Mod3/ao0:2", numberOfLines=3, numberOfSamples=2, timeout=3.0,
sampleFrequency=1000)
# write data
data = [
        [line0_sample0, line0_sample1],
        [line1_sample0, line1_sample1],
```

```
        [line2_sample0, line2_sample1]
]

aport.set(data)
```

Usage in delegation context:

```
from htf.daqmx import AnalogVoltageOutputPort

class HardwareAbstraction(DelegatorMixin):
    def __init__(self):
        self.aport = AnalogVoltageOutputPort("cDaq1Mod3/ao0:2", numberOfLines=3, numberOfSamples=2,
timeout=3.0, sampleFrequency=1000)

ha = HardwareAbstraction()
# write data
data = [
        [line0_sample0, line0_sample1],
        [line1_sample0, line1_sample1],
        [line2_sample0, line2_sample1]
]
ha.aport = data
```

```
class htf.daqmx.AnalogVoltageOutputPort(descriptor: str, minimum_voltage: float = -10.0,
maximum_voltage: float = 10.0, number_of_lines: int = 32, number_of_samples: int = 2,
sample_frequency: float = 1000.0, timeout: float = 1.0, continuous_mode: bool = False)
```

`AnalogVoltageOutputPort` is used to access a single analog output port on a DAQmx device.

A DAQmx task is automatically created and deleted in the constructor and destructor.

Parameters:

- **descriptor** – The DAQmx device and line descriptor, eg. *cDaq1Mod1/line0*
- **minimum_voltage=-10.0** – the minimum voltage.
- **maximum_voltage=10.0** – the maximum voltage.
- **number_of_lines=32** – the number of lines defined in the descriptor. For example the descriptor "cDaq1Mod3/ai0:31" defines 32 lines.
- **number_of_samples=2** – the number of samples to write for every line.
- **sample_frequency=1000.0** – the sample frequency in Hertz.
- **timeout=1.0** – the timeout when acquiring a new value in seconds.
- **continuous_mode=False** – if set to `True` continuous sampling is done and get() becomes blocking. Simply read samples in a loop.

```
close() → None
```

Close the abstractor by clearing the task.

```
get() → List[List[float]]
```

Return the last written analog values.

Returned data layout:

```
[
    [line0 sample0, line0 sample1, ..., line0 sampleN-1],
    [line1 sample0, line1 sample1, ..., line1 sampleN-1],
    ...
    [lineM-1 sample0, lineM-1 sample1, ..., lineM-1 sampleN-1]
]
```

```
set(values: List[List[float]]) → None
```

Write analog data.

Parameters:
values – samples to be written on the output port.

`values` are expected to have the following layout:

```
[
    [line0 sample0, line0 sample1, ..., line0 sampleN-1],
    [line1 sample0, line1 sample1, ..., line1 sampleN-1],
    ...
    [lineM-1 sample0, lineM-1 sample1, ..., lineM-1 sampleN-1]
]
```

Edge Counter Input

Simple usage:

```python
from htf.daqmx import EdgeCounterInput

ctr0 = EdgeCounterInput(descriptor="cDaq1Mod4/ctr0", initialCount=0, risingEdge=True, countUp=True)
# get current value
print(ctr0.get())
# reset counter
ctr0.set(0)
# or
ctr0.reset()
# set counter to 10
ctr0.set(10)
```

Usage in delegation context:

```
from htf.daqmx import EdgeCounterInput

class HardwareAbstraction(DelegatorMixin):
    def __init__(self):
        self.edges = EdgeCounterInput(descriptor="cDaq1Mod4/ctr0", initialCount=0, risingEdge=True,
countUp=True)

ha = HardwareAbstraction()
# get current value
print(ha.edges)
# reset counter
ha.edges = 0
# set counter to 10
ha.edges = 10
```

```
class htf.daqmx.EdgeCounterInput(descriptor: str, initial_count: int = 0, rising_edge: bool
= True, count_up: bool = True)
```

`EdgeCounterInput` is used to access a counter input on a DAQmx device. The counter is 32 bit wide.

A DAQmx task is automatically created and deleted in the constructor and destructor.

Parameters:

- **descriptor** (*str*) – The DAQmx device and line descriptor, eg. *cDaq1Mod1/ctr0*.
- **initial_count=0** (*int*) – The initial count.
- **rising_edge=True** (*bool*) – If set to True rising edges are counted else falling edges are counted.
- **count_up=True** (*bool*) – If set to True the counter direction is up else it is down.

```
close() → None
```

Close the abstractor by clearing the task.

```
get() → int
```

Return the current edge count.

```
reset() → None
```

Reset the counter to 0.

```
set(count: int) → None
```

Set the count to the desired value.

Parameters:
count (*int*) – the new counter value

Frequency Input

Simple usage:

```
from htf.daqmx import FrequencyInput

f0 = FrequencyInput(descriptor="cDaq1Mod4/ctr0", fMin=1000.0, fMax=10000.0, measurementTime=0.001,
risingEdge=True)
print(f0.get())
```

Usage in delegation context:

```
from htf.daqmx import FrequencyInput

class HardwareAbstraction(DelegatorMixin):
    def __init__(self):
        self.f0 = FrequencyInput(descriptor="cDaq1Mod4/ctr0", fMin=1000.0, fMax=10000.0,
measurementTime=0.001, risingEdge=True)

ha = HardwareAbstraction()
# get current value
print(ha.f0)
```

```
class htf.daqmx.FrequencyInput(descriptor: str, f_min: float, f_max: float,
measurement_time: float, rising_edge: bool = True)
```

`FrequencyInput` is used to measure a low frequency in hertz on a DAQmx device.

A DAQmx task is automatically created and deleted in the constructor and destructor.

Parameters:

- **descriptor** (*str*) – The DAQmx device and line descriptor, eg. *cDaq1Mod1/ctr0*
- **f_min** (*float*) – The minimum frequency in the input signal in Hertz.
- **f_min** – The maximum frequency in the input signal in Hertz.
- **rising_edge=True** (*bool*) – If set to True rising edges are counted else falling edges are counted.
- **measurement_time** (*float*) – The measurement time in seconds.

```
close() → None
```

Close the abstractor by clearing the task.

```
get() → float
```

Return the current frequency.

Frequency Input Inaccurate

Usage:

```
from htf.daqmx import FrequencyInputInaccurate

fi = FrequencyInputInaccurate(descriptor="cDaq1Mod4/ctr0")
# measure 10 seconds with a sample rate of 10 Hz.
frequencies = fi.sample(period=0.1, samples=100)
# output the frequency list
print("frequencies:", frequencies)
```

```
class htf.daqmx.FrequencyInputInaccurate(descriptor: str)
```

FrequencyInputInaccurate is used to measure a frequency using an *EdgeCounterInput*.

In comparison to `FrequencyInput` frequencies below 100 Hz can be measured, too.

Parameters:
descriptor – The DAQmx device and line descriptor, eg. *cDaq1Mod1/ctr0*.

```
sample(period: float = 1.0, samples: int = 1) → List[float]
```

For *samples* times measure the count of edges within one *period* and convert it to Hz.

Returns an array containing the measured frequencies.

Parameters:

- **period** – the time in seconds between two samples.
- **samples** – the amount of samples.

Returns:
the list of samples.
Return type:
list of float

Task

DAQmx is a python wrapper to the National Instruments DAQmx driver. You can use the *task* to access the driver in an object oriented way.

National Instruments does not allow to copy any documentation so you have to search C-api reference for the method name. The C-api method name always has a prefix *DAQmx*.

You need the `io` feature in your license to unlock the DAQmx features.

```
class htf.daqmx.Task(name: str)
```

`Task` automatically creates a DAQmx task.

Every method that is called does not need the prefix `DAQmx` and you must not supply the first argument if it is a taskhandle.

The task is automatically cleared when the destructor is called.

```
AddGlobalChansToTask(channelNames)
```

```
CfgAnlgEdgeRefTrig(triggerSource, triggerSlope, triggerLevel, pretriggerSamples)
```

```
CfgAnlgEdgeStartTrig(triggerSource, triggerSlope, triggerLevel)
```

```
CfgAnlgWindowRefTrig(triggerSource, triggerWhen, windowTop, windowBottom,
pretriggerSamples)
```

```
CfgAnlgWindowStartTrig(triggerSource, triggerWhen, windowTop, windowBottom)
```

```
CfgBurstHandshakingTimingExportClock(sampleMode, sampsPerChan, sampleClkRate,
sampleClkOutpTerm, sampleClkPulsePolarity, pauseWhen, readyEventActiveLevel)
```

```
CfgBurstHandshakingTimingImportClock(sampleMode, sampsPerChan, sampleClkRate,
sampleClkSrc, sampleClkActiveEdge, pauseWhen, readyEventActiveLevel)
```

```
CfgChangeDetectionTiming(risingEdgeChan, fallingEdgeChan, sampleMode, sampsPerChan)
```

```
CfgDigEdgeAdvTrig(triggerSource, triggerEdge)
```

```
CfgDigEdgeRefTrig(triggerSource, triggerEdge, pretriggerSamples)
```

```
CfgDigEdgeStartTrig(triggerSource, triggerEdge)
```

```
CfgDigPatternRefTrig(triggerSource, triggerPattern, triggerWhen, pretriggerSamples)
```

```
CfgDigPatternStartTrig(triggerSource, triggerPattern, triggerWhen)
```

```
CfgHandshakingTiming(sampleMode, sampsPerChan)
```

```
CfgImplicitTiming(sampleMode, sampsPerChan)
```

```
CfgInputBuffer(numSampsPerChan)
```

```
CfgOutputBuffer(numSampsPerChan)
```

```
CfgPipelinedSampClkTiming(source, rate, activeEdge, sampleMode, sampsPerChan)
```

```
CfgSampClkTiming(source, rate, activeEdge, sampleMode, sampsPerChan)
```

```
CfgWatchdogAOExpirStates(channelNames, expirStateArray, outputTypeArray, arraySize)
```

```
CfgWatchdogCOExpirStates(channelNames, expirStateArray, arraySize)
```

```
CfgWatchdogDOExpirStates(channelNames, expirStateArray, arraySize)
```

```
ClearTask()
```

```
ConfigureLogging(filePath, loggingMode, groupName, operation)
```

```
ControlWatchdogTask(action)
```

```
CreateAIAccelChan(physicalChannel, nameToAssignToChannel, terminalConfig, minVal,
maxVal, units, sensitivity, sensitivityUnits, currentExcitSource, currentExcitVal,
customScaleName)
```

```
CreateAIBridgeChan(physicalChannel, nameToAssignToChannel, minVal, maxVal, units,
bridgeConfig, voltageExcitSource, voltageExcitVal, nominalBridgeResistance,
customScaleName)
```

```
CreateAICurrentChan(physicalChannel, nameToAssignToChannel, terminalConfig, minVal,
maxVal, units, shuntResistorLoc, extShuntResistorVal, customScaleName)
```

```
CreateAICurrentRMSChan(physicalChannel, nameToAssignToChannel, terminalConfig, minVal,
maxVal, units, shuntResistorLoc, extShuntResistorVal, customScaleName)
```

```
CreateAIDeviceTempChan(physicalChannel, nameToAssignToChannel, units)
```

```
CreateAIForceBridgePolynomialChan(physicalChannel, nameToAssignToChannel, minVal,
maxVal, units, bridgeConfig, voltageExcitSource, voltageExcitVal, nominalBridgeResistance,
forwardCoeffs, numForwardCoeffs, reverseCoeffs, numReverseCoeffs, electricalUnits,
physicalUnits, customScaleName)
```

```
CreateAIForceBridgeTableChan(physicalChannel, nameToAssignToChannel, minVal, maxVal,
units, bridgeConfig, voltageExcitSource, voltageExcitVal, nominalBridgeResistance,
electricalVals, numElectricalVals, electricalUnits, physicalVals, numPhysicalVals,
physicalUnits, customScaleName)
```

```
CreateAIForceBridgeTwoPointLinChan(physicalChannel, nameToAssignToChannel, minVal,
maxVal, units, bridgeConfig, voltageExcitSource, voltageExcitVal, nominalBridgeResistance,
firstElectricalVal, secondElectricalVal, electricalUnits, firstPhysicalVal,
secondPhysicalVal, physicalUnits, customScaleName)
```

```
CreateAIForceIEPEChan(physicalChannel, nameToAssignToChannel, terminalConfig, minVal,
maxVal, units, sensitivity, sensitivityUnits, currentExcitSource, currentExcitVal,
customScaleName)
```

```
CreateAIFreqVoltageChan(physicalChannel, nameToAssignToChannel, minVal, maxVal, units,
thresholdLevel, hysteresis, customScaleName)
```

```
CreateAIMicrophoneChan(physicalChannel, nameToAssignToChannel, terminalConfig, units,
micSensitivity, maxSndPressLevel, currentExcitSource, currentExcitVal, customScaleName)
```

```
CreateAIPosEddyCurrProxProbeChan(physicalChannel, nameToAssignToChannel, minVal,
maxVal, units, sensitivity, sensitivityUnits, customScaleName)
```

```
CreateAIPosLVDTChan(physicalChannel, nameToAssignToChannel, minVal, maxVal, units,
sensitivity, sensitivityUnits, voltageExcitSource, voltageExcitVal, voltageExcitFreq,
ACExcitWireMode, customScaleName)
```

```
CreateAIPosRVDTChan(physicalChannel, nameToAssignToChannel, minVal, maxVal, units,
sensitivity, sensitivityUnits, voltageExcitSource, voltageExcitVal, voltageExcitFreq,
ACExcitWireMode, customScaleName)
```

```
CreateAIPressureBridgePolynomialChan(physicalChannel, nameToAssignToChannel, minVal,
maxVal, units, bridgeConfig, voltageExcitSource, voltageExcitVal, nominalBridgeResistance,
forwardCoeffs, numForwardCoeffs, reverseCoeffs, numReverseCoeffs, electricalUnits,
physicalUnits, customScaleName)
```

```
CreateAIPressureBridgeTableChan(physicalChannel, nameToAssignToChannel, minVal, maxVal,
units, bridgeConfig, voltageExcitSource, voltageExcitVal, nominalBridgeResistance,
electricalVals, numElectricalVals, electricalUnits, physicalVals, numPhysicalVals,
physicalUnits, customScaleName)
```

```
CreateAIPressureBridgeTwoPointLinChan(physicalChannel, nameToAssignToChannel, minVal,
maxVal, units, bridgeConfig, voltageExcitSource, voltageExcitVal, nominalBridgeResistance,
firstElectricalVal, secondElectricalVal, electricalUnits, firstPhysicalVal,
secondPhysicalVal, physicalUnits, customScaleName)
```

```
CreateAIRTDChan(physicalChannel, nameToAssignToChannel, minVal, maxVal, units, rtdType,
resistanceConfig, currentExcitSource, currentExcitVal, r0)
```

```
CreateAIResistanceChan(physicalChannel, nameToAssignToChannel, minVal, maxVal, units,
resistanceConfig, currentExcitSource, currentExcitVal, customScaleName)
```

```
CreateAIRosetteStrainGageChan(physicalChannel, nameToAssignToChannel, minVal, maxVal,
rosetteType, gageOrientation, rosetteMeasTypes, numRosetteMeasTypes, strainConfig,
voltageExcitSource, voltageExcitVal, gageFactor, nominalGageResistance, poissonRatio,
leadWireResistance)
```

```
CreateAIStrainGageChan(physicalChannel, nameToAssignToChannel, minVal, maxVal, units,
strainConfig, voltageExcitSource, voltageExcitVal, gageFactor, initialBridgeVoltage,
nominalGageResistance, poissonRatio, leadWireResistance, customScaleName)
```

```
CreateAITempBuiltInSensorChan(physicalChannel, nameToAssignToChannel, units)
```

```
CreateAIThrmcplChan(physicalChannel, nameToAssignToChannel, minVal, maxVal, units,
thermocoupleType, cjcSource, cjcVal, cjcChannel)
```

```
CreateAIThrmstrChanIex(physicalChannel, nameToAssignToChannel, minVal, maxVal, units,
resistanceConfig, currentExcitSource, currentExcitVal, a, b, c)
```

```
CreateAIThrmstrChanVex(physicalChannel, nameToAssignToChannel, minVal, maxVal, units,
resistanceConfig, voltageExcitSource, voltageExcitVal, a, b, c, r1)
```

```
CreateAITorqueBridgePolynomialChan(physicalChannel, nameToAssignToChannel, minVal,
maxVal, units, bridgeConfig, voltageExcitSource, voltageExcitVal, nominalBridgeResistance,
forwardCoeffs, numForwardCoeffs, reverseCoeffs, numReverseCoeffs, electricalUnits,
physicalUnits, customScaleName)
```

```
CreateAITorqueBridgeTableChan(physicalChannel, nameToAssignToChannel, minVal, maxVal,
units, bridgeConfig, voltageExcitSource, voltageExcitVal, nominalBridgeResistance,
electricalVals, numElectricalVals, electricalUnits, physicalVals, numPhysicalVals,
physicalUnits, customScaleName)
```

```
CreateAITorqueBridgeTwoPointLinChan(physicalChannel, nameToAssignToChannel, minVal,
maxVal, units, bridgeConfig, voltageExcitSource, voltageExcitVal, nominalBridgeResistance,
```

```
firstElectricalVal, secondElectricalVal, electricalUnits, firstPhysicalVal,
secondPhysicalVal, physicalUnits, customScaleName)
```

```
CreateAIVelocityIEPEChan(physicalChannel, nameToAssignToChannel, terminalConfig,
minVal, maxVal, units, sensitivity, sensitivityUnits, currentExcitSource, currentExcitVal,
customScaleName)
```

```
CreateAIVoltageChan(physicalChannel, nameToAssignToChannel, terminalConfig, minVal,
maxVal, units, customScaleName)
```

```
CreateAIVoltageChanWithExcit(physicalChannel, nameToAssignToChannel, terminalConfig,
minVal, maxVal, units, bridgeConfig, voltageExcitSource, voltageExcitVal,
useExcitForScaling, customScaleName)
```

```
CreateAIVoltageRMSChan(physicalChannel, nameToAssignToChannel, terminalConfig, minVal,
maxVal, units, customScaleName)
```

```
CreateAOCurrentChan(physicalChannel, nameToAssignToChannel, minVal, maxVal, units,
customScaleName)
```

```
CreateAOFuncGenChan(physicalChannel, nameToAssignToChannel, type, freq, amplitude,
offset)
```

```
CreateAOVoltageChan(physicalChannel, nameToAssignToChannel, minVal, maxVal, units,
customScaleName)
```

```
CreateCIAngEncoderChan(counter, nameToAssignToChannel, decodingType, ZidxEnable,
ZidxVal, ZidxPhase, units, pulsesPerRev, initialAngle, customScaleName)
```

```
CreateCICountEdgesChan(counter, nameToAssignToChannel, edge, initialCount,
countDirection)
```

```
CreateCIFreqChan(counter, nameToAssignToChannel, minVal, maxVal, units, edge,
measMethod, measTime, divisor, customScaleName)
```

```
CreateCIGPSTimestampChan(counter, nameToAssignToChannel, units, syncMethod,
customScaleName)
```

```
CreateCILinEncoderChan(counter, nameToAssignToChannel, decodingType, ZidxEnable,
ZidxVal, ZidxPhase, units, distPerPulse, initialPos, customScaleName)
```

```
CreateCIPeriodChan(counter, nameToAssignToChannel, minVal, maxVal, units, edge,
measMethod, measTime, divisor, customScaleName)
```

```
CreateCIPulseChanFreq(counter, nameToAssignToChannel, minVal, maxVal, units)
```

```
CreateCIPulseChanTicks(counter, nameToAssignToChannel, sourceTerminal, minVal, maxVal)
```

```
CreateCIPulseChanTime(counter, nameToAssignToChannel, minVal, maxVal, units)
```

```
CreateCIPulseWidthChan(counter, nameToAssignToChannel, minVal, maxVal, units,
startingEdge, customScaleName)
```

```
CreateCISemiPeriodChan(counter, nameToAssignToChannel, minVal, maxVal, units,
customScaleName)
```

```
CreateCITwoEdgeSepChan(counter, nameToAssignToChannel, minVal, maxVal, units, firstEdge,
secondEdge, customScaleName)
```

```
CreateCOPulseChanFreq(counter, nameToAssignToChannel, units, idleState, initialDelay,
freq, dutyCycle)
```

```
CreateCOPulseChanTicks(counter, nameToAssignToChannel, sourceTerminal, idleState,
initialDelay, lowTicks, highTicks)
```

```
CreateCOPulseChanTime(counter, nameToAssignToChannel, units, idleState, initialDelay,
lowTime, highTime)
```

```
CreateDIChan(lines, nameToAssignToLines, lineGrouping)
```

```
CreateDOChan(lines, nameToAssignToLines, lineGrouping)
```

```
CreateTEDSAIAccelChan(physicalChannel, nameToAssignToChannel, terminalConfig, minVal,
maxVal, units, currentExcitSource, currentExcitVal, customScaleName)
```

```
CreateTEDSAIBridgeChan(physicalChannel, nameToAssignToChannel, minVal, maxVal, units,
voltageExcitSource, voltageExcitVal, customScaleName)
```

```
CreateTEDSAICurrentChan(physicalChannel, nameToAssignToChannel, terminalConfig, minVal,
maxVal, units, shuntResistorLoc, extShuntResistorVal, customScaleName)
```

```
CreateTEDSAIForceBridgeChan(physicalChannel, nameToAssignToChannel, minVal, maxVal,
units, voltageExcitSource, voltageExcitVal, customScaleName)
```

```
CreateTEDSAIForceIEPEChan(physicalChannel, nameToAssignToChannel, terminalConfig,
minVal, maxVal, units, currentExcitSource, currentExcitVal, customScaleName)
```

```
CreateTEDSAIMicrophoneChan(physicalChannel, nameToAssignToChannel, terminalConfig,
units, maxSndPressLevel, currentExcitSource, currentExcitVal, customScaleName)
```

```
CreateTEDSAIPosLVDTChan(physicalChannel, nameToAssignToChannel, minVal, maxVal, units,
voltageExcitSource, voltageExcitVal, voltageExcitFreq, ACExcitWireMode, customScaleName)
```

```
CreateTEDSAIPosRVDTChan(physicalChannel, nameToAssignToChannel, minVal, maxVal, units,
voltageExcitSource, voltageExcitVal, voltageExcitFreq, ACExcitWireMode, customScaleName)
```

```
CreateTEDSAIPressureBridgeChan(physicalChannel, nameToAssignToChannel, minVal, maxVal,
units, voltageExcitSource, voltageExcitVal, customScaleName)
```

```
CreateTEDSAIRTDChan(physicalChannel, nameToAssignToChannel, minVal, maxVal, units,
resistanceConfig, currentExcitSource, currentExcitVal)
```

```
CreateTEDSAIResistanceChan(physicalChannel, nameToAssignToChannel, minVal, maxVal,
units, resistanceConfig, currentExcitSource, currentExcitVal, customScaleName)
```

```
CreateTEDSAIStrainGageChan(physicalChannel, nameToAssignToChannel, minVal, maxVal,
units, voltageExcitSource, voltageExcitVal, initialBridgeVoltage, leadWireResistance,
customScaleName)
```

```
CreateTEDSAIThrmcplChan(physicalChannel, nameToAssignToChannel, minVal, maxVal, units,
cjcSource, cjcVal, cjcChannel)
```

```
CreateTEDSAIThrmstrChanIex(physicalChannel, nameToAssignToChannel, minVal, maxVal,
units, resistanceConfig, currentExcitSource, currentExcitVal)
```

```
CreateTEDSAIThrmstrChanVex(physicalChannel, nameToAssignToChannel, minVal, maxVal,
units, resistanceConfig, voltageExcitSource, voltageExcitVal, r1)
```

```
CreateTEDSAITorqueBridgeChan(physicalChannel, nameToAssignToChannel, minVal, maxVal,
units, voltageExcitSource, voltageExcitVal, customScaleName)
```

```
CreateTEDSAIVoltageChan(physicalChannel, nameToAssignToChannel, terminalConfig, minVal,
maxVal, units, customScaleName)
```

```
CreateTEDSAIVoltageChanWithExcit(physicalChannel, nameToAssignToChannel,
terminalConfig, minVal, maxVal, units, voltageExcitSource, voltageExcitVal,
customScaleName)
```

```
DisableAdvTrig()
```

```
DisableRefTrig()
```

```
DisableStartTrig()
```

```
ExportSignal(signalID, outputTerminal)
```

```
GetAIACExcitFreq(channel, data)
```

```
GetAIACExcitSyncEnable(channel, data)
```

```
GetAIACExcitWireMode(channel, data)
```

```
GetAIADCCustomTimingMode(channel, data)
```

```
GetAIADCTimingMode(channel, data)
```

```
GetAIAccelSensitivity(channel, data)
```

```
GetAIAccelSensitivityUnits(channel, data)
```

```
GetAIAccelUnits(channel, data)
```

```
GetAIAcceldBRef(channel, data)
```

```
GetAIAtten(channel, data)
```

```
GetAIAutoZeroMode(channel, data)
```

```
GetAIAveragingWinSize(channel, data)
```

```
GetAIBridgeBalanceCoarsePot(channel, data)
```

```
GetAIBridgeBalanceFinePot(channel, data)
```

```
GetAIBridgeCfg(channel, data)
```

```
GetAIBridgeElectricalUnits(channel, data)
```

```
GetAIBridgeInitialRatio(channel, data)
```

```
GetAIBridgeInitialVoltage(channel, data)
```

```
GetAIBridgeNomResistance(channel, data)
```

```
GetAIBridgePhysicalUnits(channel, data)
```

```
GetAIBridgePolyForwardCoeff(channel, data, arraySizeInElements)
```

```
GetAIBridgePolyReverseCoeff(channel, data, arraySizeInElements)
```

```
GetAIBridgeScaleType(channel, data)
```

```
GetAIBridgeShuntCalEnable(channel, data)
```

```
GetAIBridgeShuntCalGainAdjust(channel, data)
```

```
GetAIBridgeShuntCalSelect(channel, data)
```

```
GetAIBridgeShuntCalShuntCalAActualResistance(channel, data)
```

```
GetAIBridgeShuntCalShuntCalAResistance(channel, data)
```

```
GetAIBridgeTableElectricalVals(channel, data, arraySizeInElements)
```

```
GetAIBridgeTablePhysicalVals(channel, data, arraySizeInElements)
```

```
GetAIBridgeTwoPointLinFirstElectricalVal(channel, data)
```

```
GetAIBridgeTwoPointLinFirstPhysicalVal(channel, data)
```

```
GetAIBridgeTwoPointLinSecondElectricalVal(channel, data)
```

```
GetAIBridgeTwoPointLinSecondPhysicalVal(channel, data)
```

```
GetAIBridgeUnits(channel, data)
```

```
GetAIChanCalApplyCalIfExp(channel, data)
```

```
GetAIChanCalCalDate(channelName, year, month, day, hour, minute)
```

```
GetAIChanCalDesc(channel, data, bufferSize)
```

```
GetAIChanCalEnableCal(channel, data)
```

```
GetAIChanCalExpDate(channelName, year, month, day, hour, minute)
```

```
GetAIChanCalHasValidCalInfo(channel, data)
```

```
GetAIChanCalOperatorName(channel, data, bufferSize)
```

```
GetAIChanCalPolyForwardCoeff(channel, data, arraySizeInElements)
```

```
GetAIChanCalPolyReverseCoeff(channel, data, arraySizeInElements)
```

```
GetAIChanCalScaleType(channel, data)
```

```
GetAIChanCalTablePreScaledVals(channel, data, arraySizeInElements)
```

```
GetAIChanCalTableScaledVals(channel, data, arraySizeInElements)
```

```
GetAIChanCalVerifAcqVals(channel, data, arraySizeInElements)
```

```
GetAIChanCalVerifRefVals(channel, data, arraySizeInElements)
```

```
GetAIConvActiveEdge(data)
```

```
GetAIConvActiveEdgeEx(deviceNames, data)
```

```
GetAIConvDigFltrEnable(data)
```

```
GetAIConvDigFltrEnableEx(deviceNames, data)
```

```
GetAIConvDigFltrMinPulseWidth(data)
```

```
GetAIConvDigFltrMinPulseWidthEx(deviceNames, data)
```

```
GetAIConvDigFltrTimebaseRate(data)
```

```
GetAIConvDigFltrTimebaseRateEx(deviceNames, data)
```

```
GetAIConvDigFltrTimebaseSrc(data, bufferSize)
```

```
GetAIConvDigFltrTimebaseSrcEx(deviceNames, data, bufferSize)
```

```
GetAIConvDigSyncEnable(data)
```

```
GetAIConvDigSyncEnableEx(deviceNames, data)
```

```
GetAIConvMaxRate(data)
```

```
GetAIConvMaxRateEx(deviceNames, data)
```

```
GetAIConvRate(data)
```

```
GetAIConvRateEx(deviceNames, data)
```

```
GetAIConvSrc(data, bufferSize)
```

```
GetAIConvSrcEx(deviceNames, data, bufferSize)
```

```
GetAIConvTimebaseDiv(data)
```

```
GetAIConvTimebaseDivEx(deviceNames, data)
```

```
GetAIConvTimebaseSrc(data)
```

```
GetAIConvTimebaseSrcEx(deviceNames, data)
```

```
GetAICoupling(channel, data)
```

```
GetAICurrentACRMSUnits(channel, data)
```

```
GetAICurrentShuntLoc(channel, data)
```

```
GetAICurrentShuntResistance(channel, data)
```

```
GetAICurrentUnits(channel, data)
```

```
GetAICustomScaleName(channel, data, bufferSize)
```

```
GetAIDCOffset(channel, data)
```

```
GetAIDataXferCustomThreshold(channel, data)
```

```
GetAIDataXferMech(channel, data)
```

```
GetAIDataXferReqCond(channel, data)
```

```
GetAIDevScalingCoeff(channel, data, arraySizeInElements)
```

```
GetAIDitherEnable(channel, data)
```

```
GetAIEddyCurrentProxProbeSensitivity(channel, data)
```

```
GetAIEddyCurrentProxProbeSensitivityUnits(channel, data)
```

```
GetAIEddyCurrentProxProbeUnits(channel, data)
```

```
GetAIEnhancedAliasRejectionEnable(channel, data)
```

```
GetAIExcitActualVal(channel, data)
```

```
GetAIExcitDCorAC(channel, data)
```

```
GetAIExcitSrc(channel, data)
```

```
GetAIExcitUseForScaling(channel, data)
```

```
GetAIExcitUseMultiplexed(channel, data)
```

```
GetAIExcitVal(channel, data)
```

```
GetAIExcitVoltageOrCurrent(channel, data)
```

```
GetAIFilterDelay(channel, data)
```

```
GetAIForceIEPESensorSensitivity(channel, data)
```

```
GetAIForceIEPESensorSensitivityUnits(channel, data)
```

```
GetAIForceReadFromChan(channel, data)
```

```
GetAIForceUnits(channel, data)
```

```
GetAIFreqHyst(channel, data)
```

```
GetAIFreqThreshVoltage(channel, data)
```

```
GetAIFreqUnits(channel, data)
```

```
GetAIGain(channel, data)
```

```
GetAIImpedance(channel, data)
```

```
GetAIInputSrc(channel, data, bufferSize)
```

```
GetAIIsTEDS(channel, data)
```

```
GetAILVDTSensitivity(channel, data)
```

```
GetAILVDTSensitivityUnits(channel, data)
```

```
GetAILVDTUnits(channel, data)
```

```
GetAILeadWireResistance(channel, data)
```

```
GetAILossyLSBRemovalCompressedSampSize(channel, data)
```

```
GetAILowpassCutoffFreq(channel, data)
```

```
GetAILowpassEnable(channel, data)
```

```
GetAILowpassSwitchCapClkSrc(channel, data)
```

```
GetAILowpassSwitchCapExtClkDiv(channel, data)
```

```
GetAILowpassSwitchCapExtClkFreq(channel, data)
```

```
GetAILowpassSwitchCapOutClkDiv(channel, data)
```

```
GetAIMax(channel, data)
```

```
GetAIMeasType(channel, data)
```

```
GetAIMemMapEnable(channel, data)
```

```
GetAIMicrophoneSensitivity(channel, data)
```

```
GetAIMin(channel, data)
```

```
GetAIOpenThrmcplDetectEnable(channel, data)
```

```
GetAIPressureUnits(channel, data)
```

```
GetAIProbeAtten(channel, data)
```

```
GetAIRTDA(channel, data)
```

```
GetAIRTDB(channel, data)
```

```
GetAIRTDC(channel, data)
```

```
GetAIRTDR0(channel, data)
```

```
GetAIRTDType(channel, data)
```

```
GetAIRVDTSensitivity(channel, data)
```

```
GetAIRVDTSensitivityUnits(channel, data)
```

```
GetAIRVDTUnits(channel, data)
```

```
GetAIRawDataCompressionType(channel, data)
```

```
GetAIRawSampJustification(channel, data)
```

```
GetAIRawSampSize(channel, data)
```

```
GetAIRemoveFilterDelay(channel, data)
```

```
GetAIResistanceCfg(channel, data)
```

```
GetAIResistanceUnits(channel, data)
```

```
GetAIResolution(channel, data)
```

```
GetAIResolutionUnits(channel, data)
```

```
GetAIRngHigh(channel, data)
```

```
GetAIRngLow(channel, data)
```

```
GetAIRosetteStrainGageOrientation(channel, data)
```

```
GetAIRosetteStrainGageRosetteMeasType(channel, data)
```

```
GetAIRosetteStrainGageRosetteType(channel, data)
```

```
GetAIRosetteStrainGageStrainChans(channel, data, bufferSize)
```

```
GetAISampAndHoldEnable(channel, data)
```

```
GetAISoundPressureMaxSoundPressureLvl(channel, data)
```

```
GetAISoundPressureUnits(channel, data)
```

```
GetAISoundPressuredBRef(channel, data)
```

```
GetAIStrainGageCfg(channel, data)
```

```
GetAIStrainGageForceReadFromChan(channel, data)
```

```
GetAIStrainGageGageFactor(channel, data)
```

```
GetAIStrainGagePoissonRatio(channel, data)
```

```
GetAIStrainUnits(channel, data)
```

```
GetAITEDSUnits(channel, data, bufferSize)
```

```
GetAITempUnits(channel, data)
```

```
GetAITermCfg(channel, data)
```

```
GetAIThrmcplCJCChan(channel, data, bufferSize)
```

```
GetAIThrmcplCJCSrc(channel, data)
```

```
GetAIThrmcplCJCVal(channel, data)
```

```
GetAIThrmcplLeadOffsetVoltage(channel, data)
```

```
GetAIThrmcplScaleType(channel, data)
```

```
GetAIThrmcplType(channel, data)
```

```
GetAIThrmstrA(channel, data)
```

```
GetAIThrmstrB(channel, data)
```

```
GetAIThrmstrC(channel, data)
```

```
GetAIThrmstrR1(channel, data)
```

```
GetAITorqueUnits(channel, data)
```

```
GetAIUsbXferReqCount(channel, data)
```

```
GetAIUsbXferReqSize(channel, data)
```

```
GetAIVelocityIEPESensorSensitivity(channel, data)
```

```
GetAIVelocityIEPESensorSensitivityUnits(channel, data)
```

```
GetAIVelocityIEPESensordBRef(channel, data)
```

```
GetAIVelocityUnits(channel, data)
```

```
GetAIVoltageACRMSUnits(channel, data)
```

```
GetAIVoltageUnits(channel, data)
```

```
GetAIVoltagedBRef(channel, data)
```

```
GetAOCurrentUnits(channel, data)
```

```
GetAOCustomScaleName(channel, data, bufferSize)
```

```
GetAODACOffsetExtSrc(channel, data, bufferSize)
```

```
GetAODACOffsetSrc(channel, data)
```

```
GetAODACOffsetVal(channel, data)
```

```
GetAODACRefAllowConnToGnd(channel, data)
```

```
GetAODACRefConnToGnd(channel, data)
```

```
GetAODACRefExtSrc(channel, data, bufferSize)
```

```
GetAODACRefSrc(channel, data)
```

```
GetAODACRefVal(channel, data)
```

```
GetAODACRngHigh(channel, data)
```

```
GetAODACRngLow(channel, data)
```

```
GetAODataXferMech(channel, data)
```

```
GetAODataXferReqCond(channel, data)
```

```
GetAODevScalingCoeff(channel, data, arraySizeInElements)
```

```
GetAOEnhancedImageRejectionEnable(channel, data)
```

```
GetAOFuncGenAmplitude(channel, data)
```

```
GetAOFuncGenFMDeviation(channel, data)
```

```
GetAOFuncGenFreq(channel, data)
```

```
GetAOFuncGenModulationType(channel, data)
```

```
GetAOFuncGenOffset(channel, data)
```

```
GetAOFuncGenSquareDutyCycle(channel, data)
```

```
GetAOFuncGenType(channel, data)
```

```
GetAOGain(channel, data)
```

```
GetAOIdleOutputBehavior(channel, data)
```

```
GetAOLoadImpedance(channel, data)
```

```
GetAOMax(channel, data)
```

```
GetAOMemMapEnable(channel, data)
```

```
GetAOMin(channel, data)
```

```
GetAOOutputImpedance(channel, data)
```

```
GetAOOutputType(channel, data)
```

```
GetAOReglitchEnable(channel, data)
```

```
GetAOResolution(channel, data)
```

```
GetAOResolutionUnits(channel, data)
```

```
GetAOTermCfg(channel, data)
```

```
GetAOUsbXferReqCount(channel, data)
```

```
GetAOUsbXferReqSize(channel, data)
```

```
GetAOUseOnlyOnBrdMem(channel, data)
```

```
GetAOVoltageCurrentLimit(channel, data)
```

```
GetAOVoltageUnits(channel, data)
```

```
GetAdvTrigType(data)
```

```
GetAnlgEdgeRefTrigCoupling(data)
```

```
GetAnlgEdgeRefTrigDigFltrEnable(data)
```

```
GetAnlgEdgeRefTrigDigFltrMinPulseWidth(data)
```

```
GetAnlgEdgeRefTrigDigFltrTimebaseRate(data)
```

```
GetAnlgEdgeRefTrigDigFltrTimebaseSrc(data, bufferSize)
```

```
GetAnlgEdgeRefTrigDigSyncEnable(data)
```

```
GetAnlgEdgeRefTrigHyst(data)
```

```
GetAnlgEdgeRefTrigLvl(data)
```

```
GetAnlgEdgeRefTrigSlope(data)
```

```
GetAnlgEdgeRefTrigSrc(data, bufferSize)
```

```
GetAnlgEdgeStartTrigCoupling(data)
```

```
GetAnlgEdgeStartTrigDigFltrEnable(data)
```

```
GetAnlgEdgeStartTrigDigFltrMinPulseWidth(data)
```

```
GetAnlgEdgeStartTrigDigFltrTimebaseRate(data)
```

```
GetAnlgEdgeStartTrigDigFltrTimebaseSrc(data, bufferSize)
```

```
GetAnlgEdgeStartTrigDigSyncEnable(data)
```

```
GetAnlgEdgeStartTrigHyst(data)
```

```
GetAnlgEdgeStartTrigLvl(data)
```

```
GetAnlgEdgeStartTrigSlope(data)
```

```
GetAnlgEdgeStartTrigSrc(data, bufferSize)
```

```
GetAnlgLvlPauseTrigCoupling(data)
```

```
GetAnlgLvlPauseTrigDigFltrEnable(data)
```

```
GetAnlgLvlPauseTrigDigFltrMinPulseWidth(data)
```

```
GetAnlgLvlPauseTrigDigFltrTimebaseRate(data)
```

```
GetAnlgLvlPauseTrigDigFltrTimebaseSrc(data, bufferSize)
```

```
GetAnlgLvlPauseTrigDigSyncEnable(data)
```

```
GetAnlgLvlPauseTrigHyst(data)
```

```
GetAnlgLvlPauseTrigLvl(data)
```

```
GetAnlgLvlPauseTrigSrc(data, bufferSize)
```

```
GetAnlgLvlPauseTrigWhen(data)
```

```
GetAnlgWinPauseTrigBtm(data)
```

```
GetAnlgWinPauseTrigCoupling(data)
```

```
GetAnlgWinPauseTrigDigFltrEnable(data)
```

```
GetAnlgWinPauseTrigDigFltrMinPulseWidth(data)
```

```
GetAnlgWinPauseTrigDigFltrTimebaseRate(data)
```

```
GetAnlgWinPauseTrigDigFltrTimebaseSrc(data, bufferSize)
```

```
GetAnlgWinPauseTrigDigSyncEnable(data)
```

```
GetAnlgWinPauseTrigSrc(data, bufferSize)
```

```
GetAnlgWinPauseTrigTop(data)
```

```
GetAnlgWinPauseTrigWhen(data)
```

```
GetAnlgWinRefTrigBtm(data)
```

```
GetAnlgWinRefTrigCoupling(data)
```

```
GetAnlgWinRefTrigDigFltrEnable(data)
```

```
GetAnlgWinRefTrigDigFltrMinPulseWidth(data)
```

```
GetAnlgWinRefTrigDigFltrTimebaseRate(data)
```

```
GetAnlgWinRefTrigDigFltrTimebaseSrc(data, bufferSize)
```

```
GetAnlgWinRefTrigDigSyncEnable(data)
```

```
GetAnlgWinRefTrigSrc(data, bufferSize)
```

```
GetAnlgWinRefTrigTop(data)
```

```
GetAnlgWinRefTrigWhen(data)
```

```
GetAnlgWinStartTrigBtm(data)
```

```
GetAnlgWinStartTrigCoupling(data)
```

```
GetAnlgWinStartTrigDigFltrEnable(data)
```

```
GetAnlgWinStartTrigDigFltrMinPulseWidth(data)
```

```
GetAnlgWinStartTrigDigFltrTimebaseRate(data)
```

```
GetAnlgWinStartTrigDigFltrTimebaseSrc(data, bufferSize)
```

```
GetAnlgWinStartTrigDigSyncEnable(data)
```

```
GetAnlgWinStartTrigSrc(data, bufferSize)
```

```
GetAnlgWinStartTrigTop(data)
```

```
GetAnlgWinStartTrigWhen(data)
```

```
GetArmStartTerm(data, bufferSize)
```

```
GetArmStartTrigType(data)
```

```
GetBufInputBufSize(data)
```

```
GetBufInputOnbrdBufSize(data)
```

```
GetBufOutputBufSize(data)
```

```
GetBufOutputOnbrdBufSize(data)
```

```
GetBufferAttribute(attribute, value)
```

```
GetCIAngEncoderInitialAngle(channel, data)
```

```
GetCIAngEncoderPulsesPerRev(channel, data)
```

```
GetCIAngEncoderUnits(channel, data)
```

```
GetCICount(channel, data)
```

```
GetCICountEdgesActiveEdge(channel, data)
```

```
GetCICountEdgesCountDirDigFltrEnable(channel, data)
```

```
GetCICountEdgesCountDirDigFltrMinPulseWidth(channel, data)
```

```
GetCICountEdgesCountDirDigFltrTimebaseRate(channel, data)
```

```
GetCICountEdgesCountDirDigFltrTimebaseSrc(channel, data, bufferSize)
```

```
GetCICountEdgesCountDirDigSyncEnable(channel, data)
```

```
GetCICountEdgesCountResetActiveEdge(channel, data)
```

```
GetCICountEdgesCountResetDigFltrEnable(channel, data)
```

```
GetCICountEdgesCountResetDigFltrMinPulseWidth(channel, data)
```

```
GetCICountEdgesCountResetDigFltrTimebaseRate(channel, data)
```

```
GetCICountEdgesCountResetDigFltrTimebaseSrc(channel, data, bufferSize)
```

```
GetCICountEdgesCountResetDigSyncEnable(channel, data)
```

```
GetCICountEdgesCountResetEnable(channel, data)
```

```
GetCICountEdgesCountResetResetCount(channel, data)
```

```
GetCICountEdgesCountResetTerm(channel, data, bufferSize)
```

```
GetCICountEdgesDigFltrEnable(channel, data)
```

```
GetCICountEdgesDigFltrMinPulseWidth(channel, data)
```

```
GetCICountEdgesDigFltrTimebaseRate(channel, data)
```

```
GetCICountEdgesDigFltrTimebaseSrc(channel, data, bufferSize)
```

```
GetCICountEdgesDigSyncEnable(channel, data)
```

```
GetCICountEdgesDir(channel, data)
```

```
GetCICountEdgesDirTerm(channel, data, bufferSize)
```

```
GetCICountEdgesInitialCnt(channel, data)
```

```
GetCICountEdgesTerm(channel, data, bufferSize)
```

```
GetCICtrTimebaseActiveEdge(channel, data)
```

```
GetCICtrTimebaseDigFltrEnable(channel, data)
```

```
GetCICtrTimebaseDigFltrMinPulseWidth(channel, data)
```

```
GetCICtrTimebaseDigFltrTimebaseRate(channel, data)
```

```
GetCICtrTimebaseDigFltrTimebaseSrc(channel, data, bufferSize)
```

```
GetCICtrTimebaseDigSyncEnable(channel, data)
```

```
GetCICtrTimebaseMasterTimebaseDiv(channel, data)
```

```
GetCICtrTimebaseRate(channel, data)
```

```
GetCICtrTimebaseSrc(channel, data, bufferSize)
```

```
GetCICustomScaleName(channel, data, bufferSize)
```

```
GetCIDataXferMech(channel, data)
```

```
GetCIDataXferReqCond(channel, data)
```

```
GetCIDupCountPrevent(channel, data)
```

```
GetCIEncoderAInputDigFltrEnable(channel, data)
```

```
GetCIEncoderAInputDigFltrMinPulseWidth(channel, data)
```

```
GetCIEncoderAInputDigFltrTimebaseRate(channel, data)
```

```
GetCIEncoderAInputDigFltrTimebaseSrc(channel, data, bufferSize)
```

```
GetCIEncoderAInputDigSyncEnable(channel, data)
```

```
GetCIEncoderAInputTerm(channel, data, bufferSize)
```

```
GetCIEncoderBInputDigFltrEnable(channel, data)
```

```
GetCIEncoderBInputDigFltrMinPulseWidth(channel, data)
```

```
GetCIEncoderBInputDigFltrTimebaseRate(channel, data)
```

```
GetCIEncoderBInputDigFltrTimebaseSrc(channel, data, bufferSize)
```

```
GetCIEncoderBInputDigSyncEnable(channel, data)
```

```
GetCIEncoderBInputTerm(channel, data, bufferSize)
```

```
GetCIEncoderDecodingType(channel, data)
```

```
GetCIEncoderZIndexEnable(channel, data)
```

```
GetCIEncoderZIndexPhase(channel, data)
```

```
GetCIEncoderZIndexVal(channel, data)
```

```
GetCIEncoderZInputDigFltrEnable(channel, data)
```

```
GetCIEncoderZInputDigFltrMinPulseWidth(channel, data)
```

```
GetCIEncoderZInputDigFltrTimebaseRate(channel, data)
```

```
GetCIEncoderZInputDigFltrTimebaseSrc(channel, data, bufferSize)
```

```
GetCIEncoderZInputDigSyncEnable(channel, data)
```

```
GetCIEncoderZInputTerm(channel, data, bufferSize)
```

```
GetCIFreqDigFltrEnable(channel, data)
```

```
GetCIFreqDigFltrMinPulseWidth(channel, data)
```

```
GetCIFreqDigFltrTimebaseRate(channel, data)
```

```
GetCIFreqDigFltrTimebaseSrc(channel, data, bufferSize)
```

```
GetCIFreqDigSyncEnable(channel, data)
```

```
GetCIFreqDiv(channel, data)
```

```
GetCIFreqEnableAveraging(channel, data)
```

```
GetCIFreqMeasMeth(channel, data)
```

```
GetCIFreqMeasTime(channel, data)
```

```
GetCIFreqStartingEdge(channel, data)
```

```
GetCIFreqTerm(channel, data, bufferSize)
```

```
GetCIFreqUnits(channel, data)
```

```
GetCIGPSSyncMethod(channel, data)
```

```
GetCIGPSSyncSrc(channel, data, bufferSize)
```

```
GetCILinEncoderDistPerPulse(channel, data)
```

```
GetCILinEncoderInitialPos(channel, data)
```

```
GetCILinEncoderUnits(channel, data)
```

```
GetCIMax(channel, data)
```

```
GetCIMeasType(channel, data)
```

```
GetCIMemMapEnable(channel, data)
```

```
GetCIMin(channel, data)
```

```
GetCINumPossiblyInvalidSamps(channel, data)
```

```
GetCIOutputState(channel, data)
```

```
GetCIPeriodDigFltrEnable(channel, data)
```

```
GetCIPeriodDigFltrMinPulseWidth(channel, data)
```

```
GetCIPeriodDigFltrTimebaseRate(channel, data)
```

```
GetCIPeriodDigFltrTimebaseSrc(channel, data, bufferSize)
```

```
GetCIPeriodDigSyncEnable(channel, data)
```

```
GetCIPeriodDiv(channel, data)
```

```
GetCIPeriodEnableAveraging(channel, data)
```

```
GetCIPeriodMeasMeth(channel, data)
```

```
GetCIPeriodMeasTime(channel, data)
```

```
GetCIPeriodStartingEdge(channel, data)
```

```
GetCIPeriodTerm(channel, data, bufferSize)
```

```
GetCIPeriodUnits(channel, data)
```

```
GetCIPrescaler(channel, data)
```

```
GetCIPulseFreqDigFltrEnable(channel, data)
```

```
GetCIPulseFreqDigFltrMinPulseWidth(channel, data)
```

```
GetCIPulseFreqDigFltrTimebaseRate(channel, data)
```

```
GetCIPulseFreqDigFltrTimebaseSrc(channel, data, bufferSize)
```

```
GetCIPulseFreqDigSyncEnable(channel, data)
```

```
GetCIPulseFreqStartEdge(channel, data)
```

```
GetCIPulseFreqTerm(channel, data, bufferSize)
```

```
GetCIPulseFreqUnits(channel, data)
```

```
GetCIPulseTicksDigFltrEnable(channel, data)
```

```
GetCIPulseTicksDigFltrMinPulseWidth(channel, data)
```

```
GetCIPulseTicksDigFltrTimebaseRate(channel, data)
```

```
GetCIPulseTicksDigFltrTimebaseSrc(channel, data, bufferSize)
```

```
GetCIPulseTicksDigSyncEnable(channel, data)
```

```
GetCIPulseTicksStartEdge(channel, data)
```

```
GetCIPulseTicksTerm(channel, data, bufferSize)
```

```
GetCIPulseTimeDigFltrEnable(channel, data)
```

```
GetCIPulseTimeDigFltrMinPulseWidth(channel, data)
```

```
GetCIPulseTimeDigFltrTimebaseRate(channel, data)
```

```
GetCIPulseTimeDigFltrTimebaseSrc(channel, data, bufferSize)
```

```
GetCIPulseTimeDigSyncEnable(channel, data)
```

```
GetCIPulseTimeStartEdge(channel, data)
```

```
GetCIPulseTimeTerm(channel, data, bufferSize)
```

```
GetCIPulseTimeUnits(channel, data)
```

```
GetCIPulseWidthDigFltrEnable(channel, data)
```

```
GetCIPulseWidthDigFltrMinPulseWidth(channel, data)
```

```
GetCIPulseWidthDigFltrTimebaseRate(channel, data)
```

```
GetCIPulseWidthDigFltrTimebaseSrc(channel, data, bufferSize)
```

```
GetCIPulseWidthDigSyncEnable(channel, data)
```

```
GetCIPulseWidthStartingEdge(channel, data)
```

```
GetCIPulseWidthTerm(channel, data, bufferSize)
```

```
GetCIPulseWidthUnits(channel, data)
```

```
GetCISemiPeriodDigFltrEnable(channel, data)
```

```
GetCISemiPeriodDigFltrMinPulseWidth(channel, data)
```

```
GetCISemiPeriodDigFltrTimebaseRate(channel, data)
```

```
GetCISemiPeriodDigFltrTimebaseSrc(channel, data, bufferSize)
```

```
GetCISemiPeriodDigSyncEnable(channel, data)
```

```
GetCISemiPeriodStartingEdge(channel, data)
```

```
GetCISemiPeriodTerm(channel, data, bufferSize)
```

```
GetCISemiPeriodUnits(channel, data)
```

```
GetCITCReached(channel, data)
```

```
GetCITimestampInitialSeconds(channel, data)
```

```
GetCITimestampUnits(channel, data)
```

```
GetCITwoEdgeSepFirstDigFltrEnable(channel, data)
```

```
GetCITwoEdgeSepFirstDigFltrMinPulseWidth(channel, data)
```

```
GetCITwoEdgeSepFirstDigFltrTimebaseRate(channel, data)
```

```
GetCITwoEdgeSepFirstDigFltrTimebaseSrc(channel, data, bufferSize)
```

```
GetCITwoEdgeSepFirstDigSyncEnable(channel, data)
```

```
GetCITwoEdgeSepFirstEdge(channel, data)
```

```
GetCITwoEdgeSepFirstTerm(channel, data, bufferSize)
```

```
GetCITwoEdgeSepSecondDigFltrEnable(channel, data)
```

```
GetCITwoEdgeSepSecondDigFltrMinPulseWidth(channel, data)
```

```
GetCITwoEdgeSepSecondDigFltrTimebaseRate(channel, data)
```

```
GetCITwoEdgeSepSecondDigFltrTimebaseSrc(channel, data, bufferSize)
```

```
GetCITwoEdgeSepSecondDigSyncEnable(channel, data)
```

```
GetCITwoEdgeSepSecondEdge(channel, data)
```

```
GetCITwoEdgeSepSecondTerm(channel, data, bufferSize)
```

```
GetCITwoEdgeSepUnits(channel, data)
```

```
GetCIUsbXferReqCount(channel, data)
```

```
GetCIUsbXferReqSize(channel, data)
```

```
GetCOAutoIncrCnt(channel, data)
```

```
GetCOConstrainedGenMode(channel, data)
```

```
GetCOCount(channel, data)
```

```
GetCOCtrTimebaseActiveEdge(channel, data)
```

```
GetCOCtrTimebaseDigFltrEnable(channel, data)
```

```
GetCOCtrTimebaseDigFltrMinPulseWidth(channel, data)
```

```
GetCOCtrTimebaseDigFltrTimebaseRate(channel, data)
```

```
GetCOCtrTimebaseDigFltrTimebaseSrc(channel, data, bufferSize)
```

```
GetCOCtrTimebaseDigSyncEnable(channel, data)
```

```
GetCOCtrTimebaseMasterTimebaseDiv(channel, data)
```

```
GetCOCtrTimebaseRate(channel, data)
```

```
GetCOCtrTimebaseSrc(channel, data, bufferSize)
```

```
GetCODataXferMech(channel, data)
```

```
GetCODataXferReqCond(channel, data)
```

```
GetCOEnableInitialDelayOnRetrigger(channel, data)
```

```
GetCOMemMapEnable(channel, data)
```

```
GetCOOutputState(channel, data)
```

```
GetCOOutputType(channel, data)
```

```
GetCOPrescaler(channel, data)
```

```
GetCOPulseDone(channel, data)
```

```
GetCOPulseDutyCyc(channel, data)
```

```
GetCOPulseFreq(channel, data)
```

```
GetCOPulseFreqInitialDelay(channel, data)
```

```
GetCOPulseFreqUnits(channel, data)
```

```
GetCOPulseHighTicks(channel, data)
```

```
GetCOPulseHighTime(channel, data)
```

```
GetCOPulseIdleState(channel, data)
```

```
GetCOPulseLowTicks(channel, data)
```

```
GetCOPulseLowTime(channel, data)
```

```
GetCOPulseTerm(channel, data, bufferSize)
```

```
GetCOPulseTicksInitialDelay(channel, data)
```

```
GetCOPulseTimeInitialDelay(channel, data)
```

```
GetCOPulseTimeUnits(channel, data)
```

```
GetCORdyForNewVal(channel, data)
```

```
GetCOUsbXferReqCount(channel, data)
```

```
GetCOUsbXferReqSize(channel, data)
```

```
GetCOUseOnlyOnBrdMem(channel, data)
```

```
GetChanAttribute(channel, attribute, value)
```

```
GetChanDescr(channel, data, bufferSize)
```

```
GetChanIsGlobal(channel, data)
```

```
GetChanType(channel, data)
```

```
GetChangeDetectDIFallingEdgePhysicalChans(data, bufferSize)
```

```
GetChangeDetectDIRisingEdgePhysicalChans(data, bufferSize)
```

```
GetChangeDetectDITristate(data)
```

```
GetDIAcquireOn(channel, data)
```

```
GetDIDataXferMech(channel, data)
```

```
GetDIDataXferReqCond(channel, data)
```

```
GetDIDigFltrEnable(channel, data)
```

```
GetDIDigFltrEnableBusMode(channel, data)
```

```
GetDIDigFltrMinPulseWidth(channel, data)
```

```
GetDIDigFltrTimebaseRate(channel, data)
```

```
GetDIDigFltrTimebaseSrc(channel, data, bufferSize)
```

```
GetDIDigSyncEnable(channel, data)
```

```
GetDIInvertLines(channel, data)
```

```
GetDILogicFamily(channel, data)
```

```
GetDIMemMapEnable(channel, data)
```

```
GetDINumLines(channel, data)
```

```
GetDITristate(channel, data)
```

```
GetDIUsbXferReqCount(channel, data)
```

```
GetDIUsbXferReqSize(channel, data)
```

```
GetDODataXferMech(channel, data)
```

```
GetDODataXferReqCond(channel, data)
```

```
GetDOGenerateOn(channel, data)
```

```
GetDOInvertLines(channel, data)
```

```
GetDOLineStatesDoneState(channel, data)
```

```
GetDOLineStatesPausedState(channel, data)
```

```
GetDOLineStatesStartState(channel, data)
```

```
GetDOLogicFamily(channel, data)
```

```
GetDOMemMapEnable(channel, data)
```

```
GetDONumLines(channel, data)
```

```
GetDOOutputDriveType(channel, data)
```

```
GetDOOvercurrentAutoReenable(channel, data)
```

```
GetDOOvercurrentLimit(channel, data)
```

```
GetDOOvercurrentReenablePeriod(channel, data)
```

```
GetDOTristate(channel, data)
```

```
GetDOUsbXferReqCount(channel, data)
```

```
GetDOUsbXferReqSize(channel, data)
```

```
GetDOUseOnlyOnBrdMem(channel, data)
```

```
GetDelayFromSampClkDelay(data)
```

```
GetDelayFromSampClkDelayEx(deviceNames, data)
```

```
GetDelayFromSampClkDelayUnits(data)
```

```
GetDelayFromSampClkDelayUnitsEx(deviceNames, data)
```

```
GetDigEdgeAdvTrigDigFltrEnable(data)
```

```
GetDigEdgeAdvTrigEdge(data)
```

```
GetDigEdgeAdvTrigSrc(data, bufferSize)
```

```
GetDigEdgeArmStartTrigDigFltrEnable(data)
```

```
GetDigEdgeArmStartTrigDigFltrMinPulseWidth(data)
```

```
GetDigEdgeArmStartTrigDigFltrTimebaseRate(data)
```

```
GetDigEdgeArmStartTrigDigFltrTimebaseSrc(data, bufferSize)
```

```
GetDigEdgeArmStartTrigDigSyncEnable(data)
```

```
GetDigEdgeArmStartTrigEdge(data)
```

```
GetDigEdgeArmStartTrigSrc(data, bufferSize)
```

```
GetDigEdgeRefTrigDigFltrEnable(data)
```

```
GetDigEdgeRefTrigDigFltrMinPulseWidth(data)
```

```
GetDigEdgeRefTrigDigFltrTimebaseRate(data)
```

```
GetDigEdgeRefTrigDigFltrTimebaseSrc(data, bufferSize)
```

```
GetDigEdgeRefTrigDigSyncEnable(data)
```

```
GetDigEdgeRefTrigEdge(data)
```

```
GetDigEdgeRefTrigSrc(data, bufferSize)
```

```
GetDigEdgeStartTrigDigFltrEnable(data)
```

```
GetDigEdgeStartTrigDigFltrMinPulseWidth(data)
```

```
GetDigEdgeStartTrigDigFltrTimebaseRate(data)
```

```
GetDigEdgeStartTrigDigFltrTimebaseSrc(data, bufferSize)
```

```
GetDigEdgeStartTrigDigSyncEnable(data)
```

```
GetDigEdgeStartTrigEdge(data)
```

```
GetDigEdgeStartTrigSrc(data, bufferSize)
```

```
GetDigEdgeWatchdogExpirTrigEdge(data)
```

```
GetDigEdgeWatchdogExpirTrigSrc(data, bufferSize)
```

```
GetDigLvlPauseTrigDigFltrEnable(data)
```

```
GetDigLvlPauseTrigDigFltrMinPulseWidth(data)
```

```
GetDigLvlPauseTrigDigFltrTimebaseRate(data)
```

```
GetDigLvlPauseTrigDigFltrTimebaseSrc(data, bufferSize)
```

```
GetDigLvlPauseTrigDigSyncEnable(data)
```

```
GetDigLvlPauseTrigSrc(data, bufferSize)
```

```
GetDigLvlPauseTrigWhen(data)
```

```
GetDigPatternPauseTrigPattern(data, bufferSize)
```

```
GetDigPatternPauseTrigSrc(data, bufferSize)
```

```
GetDigPatternPauseTrigWhen(data)
```

```
GetDigPatternRefTrigPattern(data, bufferSize)
```

```
GetDigPatternRefTrigSrc(data, bufferSize)
```

```
GetDigPatternRefTrigWhen(data)
```

```
GetDigPatternStartTrigPattern(data, bufferSize)
```

```
GetDigPatternStartTrigSrc(data, bufferSize)
```

```
GetDigPatternStartTrigWhen(data)
```

```
GetExported10MHzRefClkOutputTerm(data, bufferSize)
```

```
GetExported20MHzTimebaseOutputTerm(data, bufferSize)
```

```
GetExportedAIConvClkOutputTerm(data, bufferSize)
```

```
GetExportedAIConvClkPulsePolarity(data)
```

```
GetExportedAIHoldCmpltEventOutputTerm(data, bufferSize)
```

```
GetExportedAIHoldCmpltEventPulsePolarity(data)
```

```
GetExportedAdvCmpltEventDelay(data)
```

```
GetExportedAdvCmpltEventOutputTerm(data, bufferSize)
```

```
GetExportedAdvCmpltEventPulsePolarity(data)
```

```
GetExportedAdvCmpltEventPulseWidth(data)
```

```
GetExportedAdvTrigOutputTerm(data, bufferSize)
```

```
GetExportedAdvTrigPulsePolarity(data)
```

```
GetExportedAdvTrigPulseWidth(data)
```

```
GetExportedAdvTrigPulseWidthUnits(data)
```

```
GetExportedChangeDetectEventOutputTerm(data, bufferSize)
```

```
GetExportedChangeDetectEventPulsePolarity(data)
```

```
GetExportedCtrOutEventOutputBehavior(data)
```

```
GetExportedCtrOutEventOutputTerm(data, bufferSize)
```

```
GetExportedCtrOutEventPulsePolarity(data)
```

```
GetExportedCtrOutEventToggleIdleState(data)
```

```
GetExportedDataActiveEventLvlActiveLvl(data)
```

```
GetExportedDataActiveEventOutputTerm(data, bufferSize)
```

```
GetExportedDividedSampClkTimebaseOutputTerm(data, bufferSize)
```

```
GetExportedHshkEventDelay(data)
```

```
GetExportedHshkEventInterlockedAssertOnStart(data)
```

```
GetExportedHshkEventInterlockedAssertedLvl(data)
```

```
GetExportedHshkEventInterlockedDeassertDelay(data)
```

```
GetExportedHshkEventOutputBehavior(data)
```

```
GetExportedHshkEventOutputTerm(data, bufferSize)
```

```
GetExportedHshkEventPulsePolarity(data)
```

```
GetExportedHshkEventPulseWidth(data)
```

```
GetExportedPauseTrigLvlActiveLvl(data)
```

```
GetExportedPauseTrigOutputTerm(data, bufferSize)
```

```
GetExportedRdyForStartEventLvlActiveLvl(data)
```

```
GetExportedRdyForStartEventOutputTerm(data, bufferSize)
```

```
GetExportedRdyForXferEventDeassertCond(data)
```

```
GetExportedRdyForXferEventDeassertCondCustomThreshold(data)
```

```
GetExportedRdyForXferEventLvlActiveLvl(data)
```

```
GetExportedRdyForXferEventOutputTerm(data, bufferSize)
```

```
GetExportedRefTrigOutputTerm(data, bufferSize)
```

```
GetExportedRefTrigPulsePolarity(data)
```

```
GetExportedSampClkDelayOffset(data)
```

```
GetExportedSampClkOutputBehavior(data)
```

```
GetExportedSampClkOutputTerm(data, bufferSize)
```

```
GetExportedSampClkPulsePolarity(data)
```

```
GetExportedSampClkTimebaseOutputTerm(data, bufferSize)
```

```
GetExportedSignalAttribute(attribute, value)
```

```
GetExportedStartTrigOutputTerm(data, bufferSize)
```

```
GetExportedStartTrigPulsePolarity(data)
```

```
GetExportedSyncPulseEventOutputTerm(data, bufferSize)
```

```
GetExportedWatchdogExpiredEventOutputTerm(data, bufferSize)
```

```
GetHshkDelayAfterXfer(data)
```

```
GetHshkSampleInputDataWhen(data)
```

```
GetHshkStartCond(data)
```

```
GetHshkTrigType(data)
```

```
GetImplicitUnderflowBehavior(data)
```

```
GetInterlockedHshkTrigAssertedLvl(data)
```

```
GetInterlockedHshkTrigSrc(data, bufferSize)
```

```
GetLoggingFilePath(data, bufferSize)
```

```
GetLoggingFilePreallocationSize(data)
```

```
GetLoggingFileWriteSize(data)
```

```
GetLoggingMode(data)
```

```
GetLoggingPause(data)
```

```
GetLoggingSampsPerFile(data)
```

```
GetLoggingTDMSGroupName(data, bufferSize)
```

```
GetLoggingTDMSOperation(data)
```

```
GetMasterTimebaseRate(data)
```

```
GetMasterTimebaseSrc(data, bufferSize)
```

```
GetNthTaskChannel(index, buffer, bufferSize)
```

```
GetNthTaskDevice(index, buffer, bufferSize)
```

```
GetNthTaskReadChannel(index, buffer, bufferSize)
```

```
GetOnDemandSimultaneousAOEnable(data)
```

```
GetPauseTrigTerm(data, bufferSize)
```

```
GetPauseTrigType(data)
```

```
GetPhysicalChanName(channel, data, bufferSize)
```

```
GetReadAccessoryInsertionOrRemovalDetected(data)
```

```
GetReadAttribute(attribute, value)
```

```
GetReadAutoStart(data)
```

```
GetReadAvailSampPerChan(data)
```

```
GetReadChangeDetectHasOverflowed(data)
```

```
GetReadChannelsToRead(data, bufferSize)
```

```
GetReadCommonModeRangeErrorChans(data, bufferSize)
```

```
GetReadCommonModeRangeErrorChansExist(data)
```

```
GetReadCurrReadPos(data)
```

```
GetReadDevsWithInsertedOrRemovedAccessories(data, bufferSize)
```

```
GetReadDigitalLinesBytesPerChan(data)
```

```
GetReadNumChans(data)
```

```
GetReadOffset(data)
```

```
GetReadOpenCurrentLoopChans(data, bufferSize)
```

```
GetReadOpenCurrentLoopChansExist(data)
```

```
GetReadOpenThrmcplChans(data, bufferSize)
```

```
GetReadOpenThrmcplChansExist(data)
```

```
GetReadOverWrite(data)
```

```
GetReadOvercurrentChans(data, bufferSize)
```

```
GetReadOvercurrentChansExist(data)
```

```
GetReadOverloadedChans(data, bufferSize)
```

```
GetReadOverloadedChansExist(data)
```

```
GetReadRawDataWidth(data)
```

```
GetReadReadAllAvailSamp(data)
```

```
GetReadRelativeTo(data)
```

```
GetReadSleepTime(data)
```

```
GetReadTotalSampPerChanAcquired(data)
```

```
GetReadWaitMode(data)
```

```
GetRealTimeAttribute(attribute, value)
```

```
GetRealTimeConvLateErrorsToWarnings(data)
```

```
GetRealTimeNumOfWarmupIters(data)
```

```
GetRealTimeReportMissedSamp(data)
```

```
GetRealTimeWaitForNextSampClkWaitMode(data)
```

```
GetRealTimeWriteRecoveryMode(data)
```

```
GetRefClkRate(data)
```

```
GetRefClkSrc(data, bufferSize)
```

```
GetRefTrigAutoTrigEnable(data)
```

```
GetRefTrigAutoTriggered(data)
```

```
GetRefTrigDelay(data)
```

```
GetRefTrigPretrigSamples(data)
```

```
GetRefTrigTerm(data, bufferSize)
```

```
GetRefTrigType(data)
```

```
GetSampClkActiveEdge(data)
```

```
GetSampClkDigFltrEnable(data)
```

```
GetSampClkDigFltrMinPulseWidth(data)
```

```
GetSampClkDigFltrTimebaseRate(data)
```

```
GetSampClkDigFltrTimebaseSrc(data, bufferSize)
```

```
GetSampClkDigSyncEnable(data)
```

```
GetSampClkMaxRate(data)
```

```
GetSampClkOverrunBehavior(data)
```

```
GetSampClkRate(data)
```

```
GetSampClkSrc(data, bufferSize)
```

```
GetSampClkTerm(data, bufferSize)
```

```
GetSampClkTimebaseActiveEdge(data)
```

```
GetSampClkTimebaseDiv(data)
```

```
GetSampClkTimebaseMasterTimebaseDiv(data)
```

```
GetSampClkTimebaseRate(data)
```

```
GetSampClkTimebaseSrc(data, bufferSize)
```

```
GetSampClkTimebaseTerm(data, bufferSize)
```

```
GetSampClkTimingResponseMode(data)
```

```
GetSampClkUnderflowBehavior(data)
```

```
GetSampQuantSampMode(data)
```

```
GetSampQuantSampPerChan(data)
```

```
GetSampTimingEngine(data)
```

```
GetSampTimingType(data)
```

```
GetStartTrigDelay(data)
```

```
GetStartTrigDelayUnits(data)
```

```
GetStartTrigRetriggerable(data)
```

```
GetStartTrigTerm(data, bufferSize)
```

```
GetStartTrigType(data)
```

```
GetSwitchScanAttribute(attribute, value)
```

```
GetSwitchScanBreakMode(data)
```

```
GetSwitchScanRepeatMode(data)
```

```
GetSwitchScanWaitingForAdv(data)
```

```
GetSyncClkInterval(data)
```

```
GetSyncPulseMinDelayToStart(data)
```

```
GetSyncPulseResetDelay(data)
```

```
GetSyncPulseResetTime(data)
```

```
GetSyncPulseSrc(data, bufferSize)
```

```
GetSyncPulseSyncTime(data)
```

```
GetSyncPulseTerm(data, bufferSize)
```

```
GetTaskAttribute(attribute, value)
```

```
GetTaskChannels(data, bufferSize)
```

```
GetTaskComplete(data)
```

```
GetTaskDevices(data, bufferSize)
```

```
GetTaskName(data, bufferSize)
```

```
GetTaskNumChans(data)
```

```
GetTaskNumDevices(data)
```

```
GetTimingAttribute(attribute, value)
```

```
GetTimingAttributeEx(deviceNames, attribute, value)
```

```
GetTrigAttribute(attribute, value)
```

```
GetTriggerSyncType(data)
```

```
GetWatchdogAOExpirState(lines, data)
```

```
GetWatchdogAOOutputType(lines, data)
```

```
GetWatchdogAttribute(lines, attribute, value)
```

```
GetWatchdogCOExpirState(lines, data)
```

```
GetWatchdogDOExpirState(lines, data)
```

```
GetWatchdogExpirTrigTrigOnNetworkConnLoss(data)
```

```
GetWatchdogExpirTrigType(data)
```

```
GetWatchdogHasExpired(data)
```

```
GetWatchdogTimeout(data)
```

```
GetWriteAccessoryInsertionOrRemovalDetected(data)
```

```
GetWriteAttribute(attribute, value)
```

```
GetWriteCurrWritePos(data)
```

```
GetWriteDevsWithInsertedOrRemovedAccessories(data, bufferSize)
```

```
GetWriteDigitalLinesBytesPerChan(data)
```

```
GetWriteNextWriteIsLast(data)
```

```
GetWriteNumChans(data)
```

```
GetWriteOffset(data)
```

```
GetWriteOpenCurrentLoopChans(data, bufferSize)
```

```
GetWriteOpenCurrentLoopChansExist(data)
```

```
GetWriteOvercurrentChans(data, bufferSize)
```

```
GetWriteOvercurrentChansExist(data)
```

```
GetWriteOvertemperatureChansExist(data)
```

```
GetWritePowerSupplyFaultChans(data, bufferSize)
```

```
GetWritePowerSupplyFaultChansExist(data)
```

```
GetWriteRawDataWidth(data)
```

```
GetWriteRegenMode(data)
```

```
GetWriteRelativeTo(data)
```

```
GetWriteSleepTime(data)
```

```
GetWriteSpaceAvail(data)
```

```
GetWriteTotalSampPerChanGenerated(data)
```

```
GetWriteWaitMode(data)
```

```
IsTaskDone(isTaskDone)
```

```
PerformBridgeOffsetNullingCal(channel)
```

```
PerformBridgeOffsetNullingCalEx(channel, skipUnsupportedChannels)
```

```
PerformBridgeShuntCal(channel, shuntResistorValue, shuntResistorLocation,
bridgeResistance, skipUnsupportedChannels)
```

```
PerformStrainShuntCal(channel, shuntResistorValue, shuntResistorLocation,
skipUnsupportedChannels)
```

```
PerformThrmcplLeadOffsetNullingCal(channel, skipUnsupportedChannels)
```

```
ReadAnalogF64(numSampsPerChan, timeout, fillMode, readArray, arraySizeInSamps,
sampsPerChanRead, reserved)
```

```
ReadAnalogScalarF64(timeout, value, reserved)
```

```
ReadBinaryI16(numSampsPerChan, timeout, fillMode, readArray, arraySizeInSamps,
sampsPerChanRead, reserved)
```

```
ReadBinaryI32(numSampsPerChan, timeout, fillMode, readArray, arraySizeInSamps,
sampsPerChanRead, reserved)
```

```
ReadBinaryU16(numSampsPerChan, timeout, fillMode, readArray, arraySizeInSamps,
sampsPerChanRead, reserved)
```

```
ReadBinaryU32(numSampsPerChan, timeout, fillMode, readArray, arraySizeInSamps,
sampsPerChanRead, reserved)
```

```
ReadCounterF64(numSampsPerChan, timeout, readArray, arraySizeInSamps, sampsPerChanRead,
reserved)
```

```
ReadCounterScalarF64(timeout, value, reserved)
```

```
ReadCounterScalarU32(timeout, value, reserved)
```

```
ReadCounterU32(numSampsPerChan, timeout, readArray, arraySizeInSamps, sampsPerChanRead,
reserved)
```

```
ReadCtrFreq(numSampsPerChan, timeout, interleaved, readArrayFrequency,
readArrayDutyCycle, arraySizeInSamps, sampsPerChanRead, reserved)
```

```
ReadCtrFreqScalar(timeout, frequency, dutyCycle, reserved)
```

```
ReadCtrTicks(numSampsPerChan, timeout, interleaved, readArrayHighTicks,
readArrayLowTicks, arraySizeInSamps, sampsPerChanRead, reserved)
```

```
ReadCtrTicksScalar(timeout, highTicks, lowTicks, reserved)
```

```
ReadCtrTime(numSampsPerChan, timeout, interleaved, readArrayHighTime, readArrayLowTime,
arraySizeInSamps, sampsPerChanRead, reserved)
```

```
ReadCtrTimeScalar(timeout, highTime, lowTime, reserved)
```

```
ReadDigitalLines(numSampsPerChan, timeout, fillMode, readArray, arraySizeInBytes,
sampsPerChanRead, numBytesPerSamp, reserved)
```

```
ReadDigitalScalarU32(timeout, value, reserved)
```

```
ReadDigitalU16(numSampsPerChan, timeout, fillMode, readArray, arraySizeInSamps,
sampsPerChanRead, reserved)
```

```
ReadDigitalU32(numSampsPerChan, timeout, fillMode, readArray, arraySizeInSamps,
sampsPerChanRead, reserved)
```

```
ReadDigitalU8(numSampsPerChan, timeout, fillMode, readArray, arraySizeInSamps,
sampsPerChanRead, reserved)
```

```
ReadRaw(numSampsPerChan, timeout, readArray, arraySizeInBytes, sampsRead,
numBytesPerSamp, reserved)
```

```
ResetAIACExcitFreq(channel)
```

```
ResetAIACExcitSyncEnable(channel)
```

```
ResetAIACExcitWireMode(channel)
```

```
ResetAIADCCustomTimingMode(channel)
```

```
ResetAIADCTimingMode(channel)
```

```
ResetAIAccelSensitivity(channel)
```

```
ResetAIAccelSensitivityUnits(channel)
```

```
ResetAIAccelUnits(channel)
```

```
ResetAIAcceldBRef(channel)
```

```
ResetAIAtten(channel)
```

```
ResetAIAutoZeroMode(channel)
```

```
ResetAIAveragingWinSize(channel)
```

```
ResetAIBridgeBalanceCoarsePot(channel)
```

```
ResetAIBridgeBalanceFinePot(channel)
```

```
ResetAIBridgeCfg(channel)
```

```
ResetAIBridgeElectricalUnits(channel)
```

```
ResetAIBridgeInitialRatio(channel)
```

```
ResetAIBridgeInitialVoltage(channel)
```

```
ResetAIBridgeNomResistance(channel)
```

```
ResetAIBridgePhysicalUnits(channel)
```

```
ResetAIBridgePolyForwardCoeff(channel)
```

```
ResetAIBridgePolyReverseCoeff(channel)
```

```
ResetAIBridgeScaleType(channel)
```

```
ResetAIBridgeShuntCalEnable(channel)
```

```
ResetAIBridgeShuntCalGainAdjust(channel)
```

```
ResetAIBridgeShuntCalSelect(channel)
```

```
ResetAIBridgeShuntCalShuntCalAActualResistance(channel)
```

```
ResetAIBridgeShuntCalShuntCalAResistance(channel)
```

```
ResetAIBridgeTableElectricalVals(channel)
```

```
ResetAIBridgeTablePhysicalVals(channel)
```

```
ResetAIBridgeTwoPointLinFirstElectricalVal(channel)
```

```
ResetAIBridgeTwoPointLinFirstPhysicalVal(channel)
```

```
ResetAIBridgeTwoPointLinSecondElectricalVal(channel)
```

```
ResetAIBridgeTwoPointLinSecondPhysicalVal(channel)
```

```
ResetAIBridgeUnits(channel)
```

```
ResetAIChanCalApplyCalIfExp(channel)
```

```
ResetAIChanCalDesc(channel)
```

```
ResetAIChanCalEnableCal(channel)
```

```
ResetAIChanCalOperatorName(channel)
```

```
ResetAIChanCalPolyForwardCoeff(channel)
```

```
ResetAIChanCalPolyReverseCoeff(channel)
```

```
ResetAIChanCalScaleType(channel)
```

```
ResetAIChanCalTablePreScaledVals(channel)
```

```
ResetAIChanCalTableScaledVals(channel)
```

```
ResetAIChanCalVerifAcqVals(channel)
```

```
ResetAIChanCalVerifRefVals(channel)
```

```
ResetAIConvActiveEdge()
```

```
ResetAIConvActiveEdgeEx(deviceNames)
```

```
ResetAIConvDigFltrEnable()
```

```
ResetAIConvDigFltrEnableEx(deviceNames)
```

```
ResetAIConvDigFltrMinPulseWidth()
```

```
ResetAIConvDigFltrMinPulseWidthEx(deviceNames)
```

```
ResetAIConvDigFltrTimebaseRate()
```

```
ResetAIConvDigFltrTimebaseRateEx(deviceNames)
```

```
ResetAIConvDigFltrTimebaseSrc()
```

```
ResetAIConvDigFltrTimebaseSrcEx(deviceNames)
```

```
ResetAIConvDigSyncEnable()
```

```
ResetAIConvDigSyncEnableEx(deviceNames)
```

```
ResetAIConvRate()
```

```
ResetAIConvRateEx(deviceNames)
```

```
ResetAIConvSrc()
```

```
ResetAIConvSrcEx(deviceNames)
```

```
ResetAIConvTimebaseDiv()
```

```
ResetAIConvTimebaseDivEx(deviceNames)
```

```
ResetAIConvTimebaseSrc()
```

```
ResetAIConvTimebaseSrcEx(deviceNames)
```

```
ResetAICoupling(channel)
```

```
ResetAICurrentACRMSUnits(channel)
```

```
ResetAICurrentShuntLoc(channel)
```

```
ResetAICurrentShuntResistance(channel)
```

```
ResetAICurrentUnits(channel)
```

```
ResetAICustomScaleName(channel)
```

```
ResetAIDCOffset(channel)
```

```
ResetAIDataXferCustomThreshold(channel)
```

```
ResetAIDataXferMech(channel)
```

```
ResetAIDataXferReqCond(channel)
```

```
ResetAIDitherEnable(channel)
```

```
ResetAIEddyCurrentProxProbeSensitivity(channel)
```

```
ResetAIEddyCurrentProxProbeSensitivityUnits(channel)
```

```
ResetAIEddyCurrentProxProbeUnits(channel)
```

```
ResetAIEnhancedAliasRejectionEnable(channel)
```

```
ResetAIExcitActualVal(channel)
```

```
ResetAIExcitDCorAC(channel)
```

```
ResetAIExcitSrc(channel)
```

```
ResetAIExcitUseForScaling(channel)
```

```
ResetAIExcitUseMultiplexed(channel)
```

```
ResetAIExcitVal(channel)
```

```
ResetAIExcitVoltageOrCurrent(channel)
```

```
ResetAIForceIEPESensorSensitivity(channel)
```

```
ResetAIForceIEPESensorSensitivityUnits(channel)
```

```
ResetAIForceReadFromChan(channel)
```

```
ResetAIForceUnits(channel)
```

```
ResetAIFreqHyst(channel)
```

```
ResetAIFreqThreshVoltage(channel)
```

```
ResetAIFreqUnits(channel)
```

```
ResetAIGain(channel)
```

```
ResetAIImpedance(channel)
```

```
ResetAIInputSrc(channel)
```

```
ResetAILVDTSensitivity(channel)
```

```
ResetAILVDTSensitivityUnits(channel)
```

```
ResetAILVDTUnits(channel)
```

```
ResetAILeadWireResistance(channel)
```

```
ResetAILossyLSBRemovalCompressedSampSize(channel)
```

```
ResetAILowpassCutoffFreq(channel)
```

```
ResetAILowpassEnable(channel)
```

```
ResetAILowpassSwitchCapClkSrc(channel)
```

```
ResetAILowpassSwitchCapExtClkDiv(channel)
```

```
ResetAILowpassSwitchCapExtClkFreq(channel)
```

```
ResetAILowpassSwitchCapOutClkDiv(channel)
```

```
ResetAIMax(channel)
```

```
ResetAIMemMapEnable(channel)
```

```
ResetAIMicrophoneSensitivity(channel)
```

```
ResetAIMin(channel)
```

```
ResetAIOpenThrmcplDetectEnable(channel)
```

```
ResetAIPressureUnits(channel)
```

```
ResetAIProbeAtten(channel)
```

```
ResetAIRTDA(channel)
```

```
ResetAIRTDB(channel)
```

```
ResetAIRTDC(channel)
```

```
ResetAIRTDR0(channel)
```

```
ResetAIRTDType(channel)
```

```
ResetAIRVDTSensitivity(channel)
```

```
ResetAIRVDTSensitivityUnits(channel)
```

```
ResetAIRVDTUnits(channel)
```

```
ResetAIRawDataCompressionType(channel)
```

```
ResetAIRemoveFilterDelay(channel)
```

```
ResetAIResistanceCfg(channel)
```

```
ResetAIResistanceUnits(channel)
```

```
ResetAIRngHigh(channel)
```

```
ResetAIRngLow(channel)
```

```
ResetAIRosetteStrainGageOrientation(channel)
```

```
ResetAIRosetteStrainGageRosetteMeasType(channel)
```

```
ResetAISampAndHoldEnable(channel)
```

```
ResetAISoundPressureMaxSoundPressureLvl(channel)
```

```
ResetAISoundPressureUnits(channel)
```

```
ResetAISoundPressuredBRef(channel)
```

```
ResetAIStrainGageCfg(channel)
```

```
ResetAIStrainGageForceReadFromChan(channel)
```

```
ResetAIStrainGageGageFactor(channel)
```

```
ResetAIStrainGagePoissonRatio(channel)
```

```
ResetAIStrainUnits(channel)
```

```
ResetAITempUnits(channel)
```

```
ResetAITermCfg(channel)
```

```
ResetAIThrmcplCJCVal(channel)
```

```
ResetAIThrmcplLeadOffsetVoltage(channel)
```

```
ResetAIThrmcplScaleType(channel)
```

```
ResetAIThrmcplType(channel)
```

```
ResetAIThrmstrA(channel)
```

```
ResetAIThrmstrB(channel)
```

```
ResetAIThrmstrC(channel)
```

```
ResetAIThrmstrR1(channel)
```

```
ResetAITorqueUnits(channel)
```

```
ResetAIUsbXferReqCount(channel)
```

```
ResetAIUsbXferReqSize(channel)
```

```
ResetAIVelocityIEPESensorSensitivity(channel)
```

```
ResetAIVelocityIEPESensorSensitivityUnits(channel)
```

```
ResetAIVelocityIEPESensordBRef(channel)
```

```
ResetAIVelocityUnits(channel)
```

```
ResetAIVoltageACRMSUnits(channel)
```

```
ResetAIVoltageUnits(channel)
```

```
ResetAIVoltagedBRef(channel)
```

```
ResetAOCurrentUnits(channel)
```

```
ResetAOCustomScaleName(channel)
```

```
ResetAODACOffsetExtSrc(channel)
```

```
ResetAODACOffsetSrc(channel)
```

```
ResetAODACOffsetVal(channel)
```

```
ResetAODACRefAllowConnToGnd(channel)
```

```
ResetAODACRefConnToGnd(channel)
```

```
ResetAODACRefExtSrc(channel)
```

```
ResetAODACRefSrc(channel)
```

```
ResetAODACRefVal(channel)
```

```
ResetAODACRngHigh(channel)
```

```
ResetAODACRngLow(channel)
```

```
ResetAODataXferMech(channel)
```

```
ResetAODataXferReqCond(channel)
```

```
ResetAOEnhancedImageRejectionEnable(channel)
```

```
ResetAOFuncGenAmplitude(channel)
```

```
ResetAOFuncGenFMDeviation(channel)
```

```
ResetAOFuncGenFreq(channel)
```

```
ResetAOFuncGenModulationType(channel)
```

```
ResetAOFuncGenOffset(channel)
```

```
ResetAOFuncGenSquareDutyCycle(channel)
```

```
ResetAOFuncGenType(channel)
```

```
ResetAOGain(channel)
```

```
ResetAOIdleOutputBehavior(channel)
```

```
ResetAOLoadImpedance(channel)
```

```
ResetAOMax(channel)
```

```
ResetAOMemMapEnable(channel)
```

```
ResetAOMin(channel)
```

```
ResetAOOutputImpedance(channel)
```

```
ResetAOReglitchEnable(channel)
```

```
ResetAOResolutionUnits(channel)
```

```
ResetAOTermCfg(channel)
```

```
ResetAOUsbXferReqCount(channel)
```

```
ResetAOUsbXferReqSize(channel)
```

```
ResetAOUseOnlyOnBrdMem(channel)
```

```
ResetAOVoltageCurrentLimit(channel)
```

```
ResetAOVoltageUnits(channel)
```

```
ResetAdvTrigType()
```

```
ResetAnlgEdgeRefTrigCoupling()
```

```
ResetAnlgEdgeRefTrigDigFltrEnable()
```

```
ResetAnlgEdgeRefTrigDigFltrMinPulseWidth()
```

```
ResetAnlgEdgeRefTrigDigFltrTimebaseRate()
```

```
ResetAnlgEdgeRefTrigDigFltrTimebaseSrc()
```

```
ResetAnlgEdgeRefTrigDigSyncEnable()
```

```
ResetAnlgEdgeRefTrigHyst()
```

```
ResetAnlgEdgeRefTrigLvl()
```

```
ResetAnlgEdgeRefTrigSlope()
```

```
ResetAnlgEdgeRefTrigSrc()
```

```
ResetAnlgEdgeStartTrigCoupling()
```

```
ResetAnlgEdgeStartTrigDigFltrEnable()
```

```
ResetAnlgEdgeStartTrigDigFltrMinPulseWidth()
```

```
ResetAnlgEdgeStartTrigDigFltrTimebaseRate()
```

```
ResetAnlgEdgeStartTrigDigFltrTimebaseSrc()
```

```
ResetAnlgEdgeStartTrigDigSyncEnable()
```

```
ResetAnlgEdgeStartTrigHyst()
```

```
ResetAnlgEdgeStartTrigLvl()
```

```
ResetAnlgEdgeStartTrigSlope()
```

```
ResetAnlgEdgeStartTrigSrc()
```

```
ResetAnlgLvlPauseTrigCoupling()
```

```
ResetAnlgLvlPauseTrigDigFltrEnable()
```

```
ResetAnlgLvlPauseTrigDigFltrMinPulseWidth()
```

```
ResetAnlgLvlPauseTrigDigFltrTimebaseRate()
```

```
ResetAnlgLvlPauseTrigDigFltrTimebaseSrc()
```

```
ResetAnlgLvlPauseTrigDigSyncEnable()
```

```
ResetAnlgLvlPauseTrigHyst()
```

```
ResetAnlgLvlPauseTrigLvl()
```

```
ResetAnlgLvlPauseTrigSrc()
```

```
ResetAnlgLvlPauseTrigWhen()
```

```
ResetAnlgWinPauseTrigBtm()
```

```
ResetAnlgWinPauseTrigCoupling()
```

```
ResetAnlgWinPauseTrigDigFltrEnable()
```

```
ResetAnlgWinPauseTrigDigFltrMinPulseWidth()
```

```
ResetAnlgWinPauseTrigDigFltrTimebaseRate()
```

```
ResetAnlgWinPauseTrigDigFltrTimebaseSrc()
```

```
ResetAnlgWinPauseTrigDigSyncEnable()
```

```
ResetAnlgWinPauseTrigSrc()
```

```
ResetAnlgWinPauseTrigTop()
```

```
ResetAnlgWinPauseTrigWhen()
```

```
ResetAnlgWinRefTrigBtm()
```

```
ResetAnlgWinRefTrigCoupling()
```

```
ResetAnlgWinRefTrigDigFltrEnable()
```

```
ResetAnlgWinRefTrigDigFltrMinPulseWidth()
```

```
ResetAnlgWinRefTrigDigFltrTimebaseRate()
```

```
ResetAnlgWinRefTrigDigFltrTimebaseSrc()
```

```
ResetAnlgWinRefTrigDigSyncEnable()
```

```
ResetAnlgWinRefTrigSrc()
```

```
ResetAnlgWinRefTrigTop()
```

```
ResetAnlgWinRefTrigWhen()
```

```
ResetAnlgWinStartTrigBtm()
```

```
ResetAnlgWinStartTrigCoupling()
```

```
ResetAnlgWinStartTrigDigFltrEnable()
```

```
ResetAnlgWinStartTrigDigFltrMinPulseWidth()
```

```
ResetAnlgWinStartTrigDigFltrTimebaseRate()
```

```
ResetAnlgWinStartTrigDigFltrTimebaseSrc()
```

```
ResetAnlgWinStartTrigDigSyncEnable()
```

```
ResetAnlgWinStartTrigSrc()
```

```
ResetAnlgWinStartTrigTop()
```

```
ResetAnlgWinStartTrigWhen()
```

```
ResetArmStartTrigType()
```

```
ResetBufInputBufSize()
```

```
ResetBufOutputBufSize()
```

```
ResetBufOutputOnbrdBufSize()
```

```
ResetBufferAttribute(attribute)
```

```
ResetCIAngEncoderInitialAngle(channel)
```

```
ResetCIAngEncoderPulsesPerRev(channel)
```

```
ResetCIAngEncoderUnits(channel)
```

```
ResetCICountEdgesActiveEdge(channel)
```

```
ResetCICountEdgesCountDirDigFltrEnable(channel)
```

```
ResetCICountEdgesCountDirDigFltrMinPulseWidth(channel)
```

```
ResetCICountEdgesCountDirDigFltrTimebaseRate(channel)
```

```
ResetCICountEdgesCountDirDigFltrTimebaseSrc(channel)
```

```
ResetCICountEdgesCountDirDigSyncEnable(channel)
```

```
ResetCICountEdgesCountResetActiveEdge(channel)
```

```
ResetCICountEdgesCountResetDigFltrEnable(channel)
```

```
ResetCICountEdgesCountResetDigFltrMinPulseWidth(channel)
```

```
ResetCICountEdgesCountResetDigFltrTimebaseRate(channel)
```

```
ResetCICountEdgesCountResetDigFltrTimebaseSrc(channel)
```

```
ResetCICountEdgesCountResetDigSyncEnable(channel)
```

```
ResetCICountEdgesCountResetEnable(channel)
```

```
ResetCICountEdgesCountResetResetCount(channel)
```

```
ResetCICountEdgesCountResetTerm(channel)
```

```
ResetCICountEdgesDigFltrEnable(channel)
```

```
ResetCICountEdgesDigFltrMinPulseWidth(channel)
```

```
ResetCICountEdgesDigFltrTimebaseRate(channel)
```

```
ResetCICountEdgesDigFltrTimebaseSrc(channel)
```

```
ResetCICountEdgesDigSyncEnable(channel)
```

```
ResetCICountEdgesDir(channel)
```

```
ResetCICountEdgesDirTerm(channel)
```

```
ResetCICountEdgesInitialCnt(channel)
```

```
ResetCICountEdgesTerm(channel)
```

```
ResetCICtrTimebaseActiveEdge(channel)
```

```
ResetCICtrTimebaseDigFltrEnable(channel)
```

```
ResetCICtrTimebaseDigFltrMinPulseWidth(channel)
```

```
ResetCICtrTimebaseDigFltrTimebaseRate(channel)
```

```
ResetCICtrTimebaseDigFltrTimebaseSrc(channel)
```

```
ResetCICtrTimebaseDigSyncEnable(channel)
```

```
ResetCICtrTimebaseMasterTimebaseDiv(channel)
```

```
ResetCICtrTimebaseRate(channel)
```

```
ResetCICtrTimebaseSrc(channel)
```

```
ResetCICustomScaleName(channel)
```

```
ResetCIDataXferMech(channel)
```

```
ResetCIDataXferReqCond(channel)
```

```
ResetCIDupCountPrevent(channel)
```

```
ResetCIEncoderAInputDigFltrEnable(channel)
```

```
ResetCIEncoderAInputDigFltrMinPulseWidth(channel)
```

```
ResetCIEncoderAInputDigFltrTimebaseRate(channel)
```

```
ResetCIEncoderAInputDigFltrTimebaseSrc(channel)
```

```
ResetCIEncoderAInputDigSyncEnable(channel)
```

```
ResetCIEncoderAInputTerm(channel)
```

```
ResetCIEncoderBInputDigFltrEnable(channel)
```

```
ResetCIEncoderBInputDigFltrMinPulseWidth(channel)
```

```
ResetCIEncoderBInputDigFltrTimebaseRate(channel)
```

```
ResetCIEncoderBInputDigFltrTimebaseSrc(channel)
```

```
ResetCIEncoderBInputDigSyncEnable(channel)
```

```
ResetCIEncoderBInputTerm(channel)
```

```
ResetCIEncoderDecodingType(channel)
```

```
ResetCIEncoderZIndexEnable(channel)
```

```
ResetCIEncoderZIndexPhase(channel)
```

```
ResetCIEncoderZIndexVal(channel)
```

```
ResetCIEncoderZInputDigFltrEnable(channel)
```

```
ResetCIEncoderZInputDigFltrMinPulseWidth(channel)
```

```
ResetCIEncoderZInputDigFltrTimebaseRate(channel)
```

```
ResetCIEncoderZInputDigFltrTimebaseSrc(channel)
```

```
ResetCIEncoderZInputDigSyncEnable(channel)
```

```
ResetCIEncoderZInputTerm(channel)
```

```
ResetCIFreqDigFltrEnable(channel)
```

```
ResetCIFreqDigFltrMinPulseWidth(channel)
```

```
ResetCIFreqDigFltrTimebaseRate(channel)
```

```
ResetCIFreqDigFltrTimebaseSrc(channel)
```

```
ResetCIFreqDigSyncEnable(channel)
```

```
ResetCIFreqDiv(channel)
```

```
ResetCIFreqEnableAveraging(channel)
```

```
ResetCIFreqMeasMeth(channel)
```

```
ResetCIFreqMeasTime(channel)
```

```
ResetCIFreqStartingEdge(channel)
```

```
ResetCIFreqTerm(channel)
```

```
ResetCIFreqUnits(channel)
```

```
ResetCIGPSSyncMethod(channel)
```

```
ResetCIGPSSyncSrc(channel)
```

```
ResetCILinEncoderDistPerPulse(channel)
```

```
ResetCILinEncoderInitialPos(channel)
```

```
ResetCILinEncoderUnits(channel)
```

```
ResetCIMax(channel)
```

```
ResetCIMemMapEnable(channel)
```

```
ResetCIMin(channel)
```

```
ResetCIPeriodDigFltrEnable(channel)
```

```
ResetCIPeriodDigFltrMinPulseWidth(channel)
```

```
ResetCIPeriodDigFltrTimebaseRate(channel)
```

```
ResetCIPeriodDigFltrTimebaseSrc(channel)
```

```
ResetCIPeriodDigSyncEnable(channel)
```

```
ResetCIPeriodDiv(channel)
```

```
ResetCIPeriodEnableAveraging(channel)
```

```
ResetCIPeriodMeasMeth(channel)
```

```
ResetCIPeriodMeasTime(channel)
```

```
ResetCIPeriodStartingEdge(channel)
```

```
ResetCIPeriodTerm(channel)
```

```
ResetCIPeriodUnits(channel)
```

```
ResetCIPrescaler(channel)
```

```
ResetCIPulseFreqDigFltrEnable(channel)
```

```
ResetCIPulseFreqDigFltrMinPulseWidth(channel)
```

```
ResetCIPulseFreqDigFltrTimebaseRate(channel)
```

```
ResetCIPulseFreqDigFltrTimebaseSrc(channel)
```

```
ResetCIPulseFreqDigSyncEnable(channel)
```

```
ResetCIPulseFreqStartEdge(channel)
```

```
ResetCIPulseFreqTerm(channel)
```

```
ResetCIPulseFreqUnits(channel)
```

```
ResetCIPulseTicksDigFltrEnable(channel)
```

```
ResetCIPulseTicksDigFltrMinPulseWidth(channel)
```

```
ResetCIPulseTicksDigFltrTimebaseRate(channel)
```

```
ResetCIPulseTicksDigFltrTimebaseSrc(channel)
```

```
ResetCIPulseTicksDigSyncEnable(channel)
```

```
ResetCIPulseTicksStartEdge(channel)
```

```
ResetCIPulseTicksTerm(channel)
```

```
ResetCIPulseTimeDigFltrEnable(channel)
```

```
ResetCIPulseTimeDigFltrMinPulseWidth(channel)
```

```
ResetCIPulseTimeDigFltrTimebaseRate(channel)
```

```
ResetCIPulseTimeDigFltrTimebaseSrc(channel)
```

```
ResetCIPulseTimeDigSyncEnable(channel)
```

```
ResetCIPulseTimeStartEdge(channel)
```

```
ResetCIPulseTimeTerm(channel)
```

```
ResetCIPulseTimeUnits(channel)
```

```
ResetCIPulseWidthDigFltrEnable(channel)
```

```
ResetCIPulseWidthDigFltrMinPulseWidth(channel)
```

```
ResetCIPulseWidthDigFltrTimebaseRate(channel)
```

```
ResetCIPulseWidthDigFltrTimebaseSrc(channel)
```

```
ResetCIPulseWidthDigSyncEnable(channel)
```

```
ResetCIPulseWidthStartingEdge(channel)
```

```
ResetCIPulseWidthTerm(channel)
```

```
ResetCIPulseWidthUnits(channel)
```

```
ResetCISemiPeriodDigFltrEnable(channel)
```

```
ResetCISemiPeriodDigFltrMinPulseWidth(channel)
```

```
ResetCISemiPeriodDigFltrTimebaseRate(channel)
```

```
ResetCISemiPeriodDigFltrTimebaseSrc(channel)
```

```
ResetCISemiPeriodDigSyncEnable(channel)
```

```
ResetCISemiPeriodStartingEdge(channel)
```

```
ResetCISemiPeriodTerm(channel)
```

```
ResetCISemiPeriodUnits(channel)
```

```
ResetCITimestampInitialSeconds(channel)
```

```
ResetCITimestampUnits(channel)
```

```
ResetCITwoEdgeSepFirstDigFltrEnable(channel)
```

```
ResetCITwoEdgeSepFirstDigFltrMinPulseWidth(channel)
```

```
ResetCITwoEdgeSepFirstDigFltrTimebaseRate(channel)
```

```
ResetCITwoEdgeSepFirstDigFltrTimebaseSrc(channel)
```

```
ResetCITwoEdgeSepFirstDigSyncEnable(channel)
```

```
ResetCITwoEdgeSepFirstEdge(channel)
```

```
ResetCITwoEdgeSepFirstTerm(channel)
```

```
ResetCITwoEdgeSepSecondDigFltrEnable(channel)
```

```
ResetCITwoEdgeSepSecondDigFltrMinPulseWidth(channel)
```

```
ResetCITwoEdgeSepSecondDigFltrTimebaseRate(channel)
```

```
ResetCITwoEdgeSepSecondDigFltrTimebaseSrc(channel)
```

```
ResetCITwoEdgeSepSecondDigSyncEnable(channel)
```

```
ResetCITwoEdgeSepSecondEdge(channel)
```

```
ResetCITwoEdgeSepSecondTerm(channel)
```

```
ResetCITwoEdgeSepUnits(channel)
```

```
ResetCIUsbXferReqCount(channel)
```

```
ResetCIUsbXferReqSize(channel)
```

```
ResetCOAutoIncrCnt(channel)
```

```
ResetCOConstrainedGenMode(channel)
```

```
ResetCOCtrTimebaseActiveEdge(channel)
```

```
ResetCOCtrTimebaseDigFltrEnable(channel)
```

```
ResetCOCtrTimebaseDigFltrMinPulseWidth(channel)
```

```
ResetCOCtrTimebaseDigFltrTimebaseRate(channel)
```

```
ResetCOCtrTimebaseDigFltrTimebaseSrc(channel)
```

```
ResetCOCtrTimebaseDigSyncEnable(channel)
```

```
ResetCOCtrTimebaseMasterTimebaseDiv(channel)
```

```
ResetCOCtrTimebaseRate(channel)
```

```
ResetCOCtrTimebaseSrc(channel)
```

```
ResetCODataXferMech(channel)
```

```
ResetCODataXferReqCond(channel)
```

```
ResetCOEnableInitialDelayOnRetrigger(channel)
```

```
ResetCOMemMapEnable(channel)
```

```
ResetCOPrescaler(channel)
```

```
ResetCOPulseDutyCyc(channel)
```

```
ResetCOPulseFreq(channel)
```

```
ResetCOPulseFreqInitialDelay(channel)
```

```
ResetCOPulseFreqUnits(channel)
```

```
ResetCOPulseHighTicks(channel)
```

```
ResetCOPulseHighTime(channel)
```

```
ResetCOPulseIdleState(channel)
```

```
ResetCOPulseLowTicks(channel)
```

```
ResetCOPulseLowTime(channel)
```

```
ResetCOPulseTerm(channel)
```

```
ResetCOPulseTicksInitialDelay(channel)
```

```
ResetCOPulseTimeInitialDelay(channel)
```

```
ResetCOPulseTimeUnits(channel)
```

```
ResetCOUsbXferReqCount(channel)
```

```
ResetCOUsbXferReqSize(channel)
```

```
ResetCOUseOnlyOnBrdMem(channel)
```

```
ResetChanAttribute(channel, attribute)
```

```
ResetChanDescr(channel)
```

```
ResetChangeDetectDIFallingEdgePhysicalChans()
```

```
ResetChangeDetectDIRisingEdgePhysicalChans()
```

```
ResetChangeDetectDITristate()
```

```
ResetDIAcquireOn(channel)
```

```
ResetDIDataXferMech(channel)
```

```
ResetDIDataXferReqCond(channel)
```

```
ResetDIDigFltrEnable(channel)
```

```
ResetDIDigFltrEnableBusMode(channel)
```

```
ResetDIDigFltrMinPulseWidth(channel)
```

```
ResetDIDigFltrTimebaseRate(channel)
```

```
ResetDIDigFltrTimebaseSrc(channel)
```

```
ResetDIDigSyncEnable(channel)
```

```
ResetDIInvertLines(channel)
```

```
ResetDILogicFamily(channel)
```

```
ResetDIMemMapEnable(channel)
```

```
ResetDITristate(channel)
```

```
ResetDIUsbXferReqCount(channel)
```

```
ResetDIUsbXferReqSize(channel)
```

```
ResetDODataXferMech(channel)
```

```
ResetDODataXferReqCond(channel)
```

```
ResetDOGenerateOn(channel)
```

```
ResetDOInvertLines(channel)
```

```
ResetDOLineStatesDoneState(channel)
```

```
ResetDOLineStatesPausedState(channel)
```

```
ResetDOLineStatesStartState(channel)
```

```
ResetDOLogicFamily(channel)
```

```
ResetDOMemMapEnable(channel)
```

```
ResetDOOutputDriveType(channel)
```

```
ResetDOOvercurrentAutoReenable(channel)
```

```
ResetDOOvercurrentLimit(channel)
```

```
ResetDOOvercurrentReenablePeriod(channel)
```

```
ResetDOTristate(channel)
```

```
ResetDOUsbXferReqCount(channel)
```

```
ResetDOUsbXferReqSize(channel)
```

```
ResetDOUseOnlyOnBrdMem(channel)
```

```
ResetDelayFromSampClkDelay()
```

```
ResetDelayFromSampClkDelayEx(deviceNames)
```

```
ResetDelayFromSampClkDelayUnits()
```

```
ResetDelayFromSampClkDelayUnitsEx(deviceNames)
```

```
ResetDigEdgeAdvTrigDigFltrEnable()
```

```
ResetDigEdgeAdvTrigEdge()
```

```
ResetDigEdgeAdvTrigSrc()
```

```
ResetDigEdgeArmStartTrigDigFltrEnable()
```

```
ResetDigEdgeArmStartTrigDigFltrMinPulseWidth()
```

```
ResetDigEdgeArmStartTrigDigFltrTimebaseRate()
```

```
ResetDigEdgeArmStartTrigDigFltrTimebaseSrc()
```

```
ResetDigEdgeArmStartTrigDigSyncEnable()
```

```
ResetDigEdgeArmStartTrigEdge()
```

```
ResetDigEdgeArmStartTrigSrc()
```

```
ResetDigEdgeRefTrigDigFltrEnable()
```

```
ResetDigEdgeRefTrigDigFltrMinPulseWidth()
```

```
ResetDigEdgeRefTrigDigFltrTimebaseRate()
```

```
ResetDigEdgeRefTrigDigFltrTimebaseSrc()
```

```
ResetDigEdgeRefTrigDigSyncEnable()
```

```
ResetDigEdgeRefTrigEdge()
```

```
ResetDigEdgeRefTrigSrc()
```

```
ResetDigEdgeStartTrigDigFltrEnable()
```

```
ResetDigEdgeStartTrigDigFltrMinPulseWidth()
```

```
ResetDigEdgeStartTrigDigFltrTimebaseRate()
```

```
ResetDigEdgeStartTrigDigFltrTimebaseSrc()
```

```
ResetDigEdgeStartTrigDigSyncEnable()
```

```
ResetDigEdgeStartTrigEdge()
```

```
ResetDigEdgeStartTrigSrc()
```

```
ResetDigEdgeWatchdogExpirTrigEdge()
```

```
ResetDigEdgeWatchdogExpirTrigSrc()
```

```
ResetDigLvlPauseTrigDigFltrEnable()
```

```
ResetDigLvlPauseTrigDigFltrMinPulseWidth()
```

```
ResetDigLvlPauseTrigDigFltrTimebaseRate()
```

```
ResetDigLvlPauseTrigDigFltrTimebaseSrc()
```

```
ResetDigLvlPauseTrigDigSyncEnable()
```

```
ResetDigLvlPauseTrigSrc()
```

```
ResetDigLvlPauseTrigWhen()
```

```
ResetDigPatternPauseTrigPattern()
```

```
ResetDigPatternPauseTrigSrc()
```

```
ResetDigPatternPauseTrigWhen()
```

```
ResetDigPatternRefTrigPattern()
```

```
ResetDigPatternRefTrigSrc()
```

```
ResetDigPatternRefTrigWhen()
```

```
ResetDigPatternStartTrigPattern()
```

```
ResetDigPatternStartTrigSrc()
```

```
ResetDigPatternStartTrigWhen()
```

```
ResetExported10MHzRefClkOutputTerm()
```

```
ResetExported20MHzTimebaseOutputTerm()
```

```
ResetExportedAIConvClkOutputTerm()
```

```
ResetExportedAIHoldCmpltEventOutputTerm()
```

```
ResetExportedAIHoldCmpltEventPulsePolarity()
```

```
ResetExportedAdvCmpltEventDelay()
```

```
ResetExportedAdvCmpltEventOutputTerm()
```

```
ResetExportedAdvCmpltEventPulsePolarity()
```

```
ResetExportedAdvCmpltEventPulseWidth()
```

```
ResetExportedAdvTrigOutputTerm()
```

```
ResetExportedAdvTrigPulseWidth()
```

```
ResetExportedAdvTrigPulseWidthUnits()
```

```
ResetExportedChangeDetectEventOutputTerm()
```

```
ResetExportedChangeDetectEventPulsePolarity()
```

```
ResetExportedCtrOutEventOutputBehavior()
```

```
ResetExportedCtrOutEventOutputTerm()
```

```
ResetExportedCtrOutEventPulsePolarity()
```

```
ResetExportedCtrOutEventToggleIdleState()
```

```
ResetExportedDataActiveEventLvlActiveLvl()
```

```
ResetExportedDataActiveEventOutputTerm()
```

```
ResetExportedDividedSampClkTimebaseOutputTerm()
```

```
ResetExportedHshkEventDelay()
```

```
ResetExportedHshkEventInterlockedAssertOnStart()
```

```
ResetExportedHshkEventInterlockedAssertedLvl()
```

```
ResetExportedHshkEventInterlockedDeassertDelay()
```

```
ResetExportedHshkEventOutputBehavior()
```

```
ResetExportedHshkEventOutputTerm()
```

```
ResetExportedHshkEventPulsePolarity()
```

```
ResetExportedHshkEventPulseWidth()
```

```
ResetExportedPauseTrigLvlActiveLvl()
```

```
ResetExportedPauseTrigOutputTerm()
```

```
ResetExportedRdyForStartEventLvlActiveLvl()
```

```
ResetExportedRdyForStartEventOutputTerm()
```

```
ResetExportedRdyForXferEventDeassertCond()
```

```
ResetExportedRdyForXferEventDeassertCondCustomThreshold()
```

```
ResetExportedRdyForXferEventLvlActiveLvl()
```

```
ResetExportedRdyForXferEventOutputTerm()
```

```
ResetExportedRefTrigOutputTerm()
```

```
ResetExportedRefTrigPulsePolarity()
```

```
ResetExportedSampClkDelayOffset()
```

```
ResetExportedSampClkOutputBehavior()
```

```
ResetExportedSampClkOutputTerm()
```

```
ResetExportedSampClkPulsePolarity()
```

```
ResetExportedSampClkTimebaseOutputTerm()
```

```
ResetExportedSignalAttribute(attribute)
```

```
ResetExportedStartTrigOutputTerm()
```

```
ResetExportedStartTrigPulsePolarity()
```

```
ResetExportedSyncPulseEventOutputTerm()
```

```
ResetExportedWatchdogExpiredEventOutputTerm()
```

```
ResetHshkDelayAfterXfer()
```

```
ResetHshkSampleInputDataWhen()
```

```
ResetHshkStartCond()
```

```
ResetHshkTrigType()
```

```
ResetImplicitUnderflowBehavior()
```

```
ResetInterlockedHshkTrigAssertedLvl()
```

```
ResetInterlockedHshkTrigSrc()
```

```
ResetLoggingFilePath()
```

```
ResetLoggingFilePreallocationSize()
```

```
ResetLoggingFileWriteSize()
```

```
ResetLoggingMode()
```

```
ResetLoggingPause()
```

```
ResetLoggingSampsPerFile()
```

```
ResetLoggingTDMSGroupName()
```

```
ResetLoggingTDMSOperation()
```

```
ResetMasterTimebaseRate()
```

```
ResetMasterTimebaseSrc()
```

```
ResetOnDemandSimultaneousAOEnable()
```

```
ResetPauseTrigType()
```

```
ResetReadAttribute(attribute)
```

```
ResetReadAutoStart()
```

```
ResetReadChannelsToRead()
```

```
ResetReadOffset()
```

```
ResetReadOverWrite()
```

```
ResetReadReadAllAvailSamp()
```

```
ResetReadRelativeTo()
```

```
ResetReadSleepTime()
```

```
ResetReadWaitMode()
```

```
ResetRealTimeAttribute(attribute)
```

```
ResetRealTimeConvLateErrorsToWarnings()
```

```
ResetRealTimeNumOfWarmupIters()
```

```
ResetRealTimeReportMissedSamp()
```

```
ResetRealTimeWaitForNextSampClkWaitMode()
```

```
ResetRealTimeWriteRecoveryMode()
```

```
ResetRefClkRate()
```

```
ResetRefClkSrc()
```

```
ResetRefTrigAutoTrigEnable()
```

```
ResetRefTrigDelay()
```

```
ResetRefTrigPretrigSamples()
```

```
ResetRefTrigType()
```

```
ResetSampClkActiveEdge()
```

```
ResetSampClkDigFltrEnable()
```

```
ResetSampClkDigFltrMinPulseWidth()
```

```
ResetSampClkDigFltrTimebaseRate()
```

```
ResetSampClkDigFltrTimebaseSrc()
```

```
ResetSampClkDigSyncEnable()
```

```
ResetSampClkOverrunBehavior()
```

```
ResetSampClkRate()
```

```
ResetSampClkSrc()
```

```
ResetSampClkTimebaseActiveEdge()
```

```
ResetSampClkTimebaseDiv()
```

```
ResetSampClkTimebaseMasterTimebaseDiv()
```

```
ResetSampClkTimebaseRate()
```

```
ResetSampClkTimebaseSrc()
```

```
ResetSampClkTimingResponseMode()
```

```
ResetSampClkUnderflowBehavior()
```

```
ResetSampQuantSampMode()
```

```
ResetSampQuantSampPerChan()
```

```
ResetSampTimingEngine()
```

```
ResetSampTimingType()
```

```
ResetStartTrigDelay()
```

```
ResetStartTrigDelayUnits()
```

```
ResetStartTrigRetriggerable()
```

```
ResetStartTrigType()
```

```
ResetSwitchScanAttribute(attribute)
```

```
ResetSwitchScanBreakMode()
```

```
ResetSwitchScanRepeatMode()
```

```
ResetSyncClkInterval()
```

```
ResetSyncPulseMinDelayToStart()
```

```
ResetSyncPulseResetDelay()
```

```
ResetSyncPulseSrc()
```

```
ResetTimingAttribute(attribute)
```

```
ResetTimingAttributeEx(deviceNames, attribute)
```

```
ResetTrigAttribute(attribute)
```

```
ResetTriggerSyncType()
```

```
ResetWatchdogAOExpirState(lines)
```

```
ResetWatchdogAOOutputType(lines)
```

```
ResetWatchdogAttribute(lines, attribute)
```

```
ResetWatchdogCOExpirState(lines)
```

```
ResetWatchdogDOExpirState(lines)
```

```
ResetWatchdogExpirTrigTrigOnNetworkConnLoss()
```

```
ResetWatchdogExpirTrigType()
```

```
ResetWatchdogTimeout()
```

```
ResetWriteAttribute(attribute)
```

```
ResetWriteNextWriteIsLast()
```

```
ResetWriteOffset()
```

```
ResetWriteRegenMode()
```

```
ResetWriteRelativeTo()
```

```
ResetWriteSleepTime()
```

```
ResetWriteWaitMode()
```

```
SaveGlobalChan(channelName, saveAs, author, options)
```

```
SaveTask(saveAs, author, options)
```

```
SendSoftwareTrigger(triggerID)
```

```
SetAIACExcitFreq(channel, data)
```

```
SetAIACExcitSyncEnable(channel, data)
```

```
SetAIACExcitWireMode(channel, data)
```

```
SetAIADCCustomTimingMode(channel, data)
```

```
SetAIADCTimingMode(channel, data)
```

```
SetAIAccelSensitivity(channel, data)
```

```
SetAIAccelSensitivityUnits(channel, data)
```

```
SetAIAccelUnits(channel, data)
```

```
SetAIAcceldBRef(channel, data)
```

```
SetAIAtten(channel, data)
```

```
SetAIAutoZeroMode(channel, data)
```

```
SetAIAveragingWinSize(channel, data)
```

```
SetAIBridgeBalanceCoarsePot(channel, data)
```

```
SetAIBridgeBalanceFinePot(channel, data)
```

```
SetAIBridgeCfg(channel, data)
```

```
SetAIBridgeElectricalUnits(channel, data)
```

```
SetAIBridgeInitialRatio(channel, data)
```

```
SetAIBridgeInitialVoltage(channel, data)
```

```
SetAIBridgeNomResistance(channel, data)
```

```
SetAIBridgePhysicalUnits(channel, data)
```

```
SetAIBridgePolyForwardCoeff(channel, data, arraySizeInElements)
```

```
SetAIBridgePolyReverseCoeff(channel, data, arraySizeInElements)
```

```
SetAIBridgeScaleType(channel, data)
```

```
SetAIBridgeShuntCalEnable(channel, data)
```

```
SetAIBridgeShuntCalGainAdjust(channel, data)
```

```
SetAIBridgeShuntCalSelect(channel, data)
```

```
SetAIBridgeShuntCalShuntCalAActualResistance(channel, data)
```

```
SetAIBridgeShuntCalShuntCalAResistance(channel, data)
```

```
SetAIBridgeTableElectricalVals(channel, data, arraySizeInElements)
```

```
SetAIBridgeTablePhysicalVals(channel, data, arraySizeInElements)
```

```
SetAIBridgeTwoPointLinFirstElectricalVal(channel, data)
```

```
SetAIBridgeTwoPointLinFirstPhysicalVal(channel, data)
```

```
SetAIBridgeTwoPointLinSecondElectricalVal(channel, data)
```

```
SetAIBridgeTwoPointLinSecondPhysicalVal(channel, data)
```

```
SetAIBridgeUnits(channel, data)
```

```
SetAIChanCalApplyCalIfExp(channel, data)
```

```
SetAIChanCalCalDate(channelName, year, month, day, hour, minute)
```

```
SetAIChanCalDesc(channel, data)
```

```
SetAIChanCalEnableCal(channel, data)
```

```
SetAIChanCalExpDate(channelName, year, month, day, hour, minute)
```

```
SetAIChanCalOperatorName(channel, data)
```

```
SetAIChanCalPolyForwardCoeff(channel, data, arraySizeInElements)
```

```
SetAIChanCalPolyReverseCoeff(channel, data, arraySizeInElements)
```

```
SetAIChanCalScaleType(channel, data)
```

```
SetAIChanCalTablePreScaledVals(channel, data, arraySizeInElements)
```

```
SetAIChanCalTableScaledVals(channel, data, arraySizeInElements)
```

```
SetAIChanCalVerifAcqVals(channel, data, arraySizeInElements)
```

```
SetAIChanCalVerifRefVals(channel, data, arraySizeInElements)
```

```
SetAIConvActiveEdge(data)
```

```
SetAIConvActiveEdgeEx(deviceNames, data)
```

```
SetAIConvDigFltrEnable(data)
```

```
SetAIConvDigFltrEnableEx(deviceNames, data)
```

```
SetAIConvDigFltrMinPulseWidth(data)
```

```
SetAIConvDigFltrMinPulseWidthEx(deviceNames, data)
```

```
SetAIConvDigFltrTimebaseRate(data)
```

```
SetAIConvDigFltrTimebaseRateEx(deviceNames, data)
```

```
SetAIConvDigFltrTimebaseSrc(data)
```

```
SetAIConvDigFltrTimebaseSrcEx(deviceNames, data)
```

```
SetAIConvDigSyncEnable(data)
```

```
SetAIConvDigSyncEnableEx(deviceNames, data)
```

```
SetAIConvRate(data)
```

```
SetAIConvRateEx(deviceNames, data)
```

```
SetAIConvSrc(data)
```

```
SetAIConvSrcEx(deviceNames, data)
```

```
SetAIConvTimebaseDiv(data)
```

```
SetAIConvTimebaseDivEx(deviceNames, data)
```

```
SetAIConvTimebaseSrc(data)
```

```
SetAIConvTimebaseSrcEx(deviceNames, data)
```

```
SetAICoupling(channel, data)
```

```
SetAICurrentACRMSUnits(channel, data)
```

```
SetAICurrentShuntLoc(channel, data)
```

```
SetAICurrentShuntResistance(channel, data)
```

```
SetAICurrentUnits(channel, data)
```

```
SetAICustomScaleName(channel, data)
```

```
SetAIDCOffset(channel, data)
```

```
SetAIDataXferCustomThreshold(channel, data)
```

```
SetAIDataXferMech(channel, data)
```

```
SetAIDataXferReqCond(channel, data)
```

```
SetAIDitherEnable(channel, data)
```

```
SetAIEddyCurrentProxProbeSensitivity(channel, data)
```

```
SetAIEddyCurrentProxProbeSensitivityUnits(channel, data)
```

```
SetAIEddyCurrentProxProbeUnits(channel, data)
```

```
SetAIEnhancedAliasRejectionEnable(channel, data)
```

```
SetAIExcitActualVal(channel, data)
```

```
SetAIExcitDCorAC(channel, data)
```

```
SetAIExcitSrc(channel, data)
```

```
SetAIExcitUseForScaling(channel, data)
```

```
SetAIExcitUseMultiplexed(channel, data)
```

```
SetAIExcitVal(channel, data)
```

```
SetAIExcitVoltageOrCurrent(channel, data)
```

```
SetAIForceIEPESensorSensitivity(channel, data)
```

```
SetAIForceIEPESensorSensitivityUnits(channel, data)
```

```
SetAIForceReadFromChan(channel, data)
```

```
SetAIForceUnits(channel, data)
```

```
SetAIFreqHyst(channel, data)
```

```
SetAIFreqThreshVoltage(channel, data)
```

```
SetAIFreqUnits(channel, data)
```

```
SetAIGain(channel, data)
```

```
SetAIImpedance(channel, data)
```

```
SetAIInputSrc(channel, data)
```

```
SetAILVDTSensitivity(channel, data)
```

```
SetAILVDTSensitivityUnits(channel, data)
```

```
SetAILVDTUnits(channel, data)
```

```
SetAILeadWireResistance(channel, data)
```

```
SetAILossyLSBRemovalCompressedSampSize(channel, data)
```

```
SetAILowpassCutoffFreq(channel, data)
```

```
SetAILowpassEnable(channel, data)
```

```
SetAILowpassSwitchCapClkSrc(channel, data)
```

```
SetAILowpassSwitchCapExtClkDiv(channel, data)
```

```
SetAILowpassSwitchCapExtClkFreq(channel, data)
```

```
SetAILowpassSwitchCapOutClkDiv(channel, data)
```

```
SetAIMax(channel, data)
```

```
SetAIMemMapEnable(channel, data)
```

```
SetAIMicrophoneSensitivity(channel, data)
```

```
SetAIMin(channel, data)
```

```
SetAIOpenThrmcplDetectEnable(channel, data)
```

```
SetAIPressureUnits(channel, data)
```

```
SetAIProbeAtten(channel, data)
```

```
SetAIRTDA(channel, data)
```

```
SetAIRTDB(channel, data)
```

```
SetAIRTDC(channel, data)
```

```
SetAIRTDR0(channel, data)
```

```
SetAIRTDType(channel, data)
```

```
SetAIRVDTSensitivity(channel, data)
```

```
SetAIRVDTSensitivityUnits(channel, data)
```

```
SetAIRVDTUnits(channel, data)
```

```
SetAIRawDataCompressionType(channel, data)
```

```
SetAIRemoveFilterDelay(channel, data)
```

```
SetAIResistanceCfg(channel, data)
```

```
SetAIResistanceUnits(channel, data)
```

```
SetAIRngHigh(channel, data)
```

```
SetAIRngLow(channel, data)
```

```
SetAIRosetteStrainGageOrientation(channel, data)
```

```
SetAIRosetteStrainGageRosetteMeasType(channel, data)
```

```
SetAISampAndHoldEnable(channel, data)
```

```
SetAISoundPressureMaxSoundPressureLvl(channel, data)
```

```
SetAISoundPressureUnits(channel, data)
```

```
SetAISoundPressuredBRef(channel, data)
```

```
SetAIStrainGageCfg(channel, data)
```

```
SetAIStrainGageForceReadFromChan(channel, data)
```

```
SetAIStrainGageGageFactor(channel, data)
```

```
SetAIStrainGagePoissonRatio(channel, data)
```

```
SetAIStrainUnits(channel, data)
```

```
SetAITempUnits(channel, data)
```

```
SetAITermCfg(channel, data)
```

```
SetAIThrmcplCJCVal(channel, data)
```

```
SetAIThrmcplLeadOffsetVoltage(channel, data)
```

```
SetAIThrmcplScaleType(channel, data)
```

```
SetAIThrmcplType(channel, data)
```

```
SetAIThrmstrA(channel, data)
```

```
SetAIThrmstrB(channel, data)
```

```
SetAIThrmstrC(channel, data)
```

```
SetAIThrmstrR1(channel, data)
```

```
SetAITorqueUnits(channel, data)
```

```
SetAIUsbXferReqCount(channel, data)
```

```
SetAIUsbXferReqSize(channel, data)
```

```
SetAIVelocityIEPESensorSensitivity(channel, data)
```

```
SetAIVelocityIEPESensorSensitivityUnits(channel, data)
```

```
SetAIVelocityIEPESensordBRef(channel, data)
```

```
SetAIVelocityUnits(channel, data)
```

```
SetAIVoltageACRMSUnits(channel, data)
```

```
SetAIVoltageUnits(channel, data)
```

```
SetAIVoltagedBRef(channel, data)
```

```
SetAOCurrentUnits(channel, data)
```

```
SetAOCustomScaleName(channel, data)
```

```
SetAODACOffsetExtSrc(channel, data)
```

```
SetAODACOffsetSrc(channel, data)
```

```
SetAODACOffsetVal(channel, data)
```

```
SetAODACRefAllowConnToGnd(channel, data)
```

```
SetAODACRefConnToGnd(channel, data)
```

```
SetAODACRefExtSrc(channel, data)
```

```
SetAODACRefSrc(channel, data)
```

```
SetAODACRefVal(channel, data)
```

```
SetAODACRngHigh(channel, data)
```

```
SetAODACRngLow(channel, data)
```

```
SetAODataXferMech(channel, data)
```

```
SetAODataXferReqCond(channel, data)
```

```
SetAOEnhancedImageRejectionEnable(channel, data)
```

```
SetAOFuncGenAmplitude(channel, data)
```

```
SetAOFuncGenFMDeviation(channel, data)
```

```
SetAOFuncGenFreq(channel, data)
```

```
SetAOFuncGenModulationType(channel, data)
```

```
SetAOFuncGenOffset(channel, data)
```

```
SetAOFuncGenSquareDutyCycle(channel, data)
```

```
SetAOFuncGenType(channel, data)
```

```
SetAOGain(channel, data)
```

```
SetAOIdleOutputBehavior(channel, data)
```

```
SetAOLoadImpedance(channel, data)
```

```
SetAOMax(channel, data)
```

```
SetAOMemMapEnable(channel, data)
```

```
SetAOMin(channel, data)
```

```
SetAOOutputImpedance(channel, data)
```

```
SetAOReglitchEnable(channel, data)
```

```
SetAOResolutionUnits(channel, data)
```

```
SetAOTermCfg(channel, data)
```

```
SetAOUsbXferReqCount(channel, data)
```

```
SetAOUsbXferReqSize(channel, data)
```

```
SetAOUseOnlyOnBrdMem(channel, data)
```

```
SetAOVoltageCurrentLimit(channel, data)
```

```
SetAOVoltageUnits(channel, data)
```

```
SetAdvTrigType(data)
```

```
SetAnlgEdgeRefTrigCoupling(data)
```

```
SetAnlgEdgeRefTrigDigFltrEnable(data)
```

```
SetAnlgEdgeRefTrigDigFltrMinPulseWidth(data)
```

```
SetAnlgEdgeRefTrigDigFltrTimebaseRate(data)
```

```
SetAnlgEdgeRefTrigDigFltrTimebaseSrc(data)
```

```
SetAnlgEdgeRefTrigDigSyncEnable(data)
```

```
SetAnlgEdgeRefTrigHyst(data)
```

```
SetAnlgEdgeRefTrigLvl(data)
```

```
SetAnlgEdgeRefTrigSlope(data)
```

```
SetAnlgEdgeRefTrigSrc(data)
```

```
SetAnlgEdgeStartTrigCoupling(data)
```

```
SetAnlgEdgeStartTrigDigFltrEnable(data)
```

```
SetAnlgEdgeStartTrigDigFltrMinPulseWidth(data)
```

```
SetAnlgEdgeStartTrigDigFltrTimebaseRate(data)
```

```
SetAnlgEdgeStartTrigDigFltrTimebaseSrc(data)
```

```
SetAnlgEdgeStartTrigDigSyncEnable(data)
```

```
SetAnlgEdgeStartTrigHyst(data)
```

```
SetAnlgEdgeStartTrigLvl(data)
```

```
SetAnlgEdgeStartTrigSlope(data)
```

```
SetAnlgEdgeStartTrigSrc(data)
```

```
SetAnlgLvlPauseTrigCoupling(data)
```

```
SetAnlgLvlPauseTrigDigFltrEnable(data)
```

```
SetAnlgLvlPauseTrigDigFltrMinPulseWidth(data)
```

```
SetAnlgLvlPauseTrigDigFltrTimebaseRate(data)
```

```
SetAnlgLvlPauseTrigDigFltrTimebaseSrc(data)
```

```
SetAnlgLvlPauseTrigDigSyncEnable(data)
```

```
SetAnlgLvlPauseTrigHyst(data)
```

```
SetAnlgLvlPauseTrigLvl(data)
```

```
SetAnlgLvlPauseTrigSrc(data)
```

```
SetAnlgLvlPauseTrigWhen(data)
```

```
SetAnlgWinPauseTrigBtm(data)
```

```
SetAnlgWinPauseTrigCoupling(data)
```

```
SetAnlgWinPauseTrigDigFltrEnable(data)
```

```
SetAnlgWinPauseTrigDigFltrMinPulseWidth(data)
```

```
SetAnlgWinPauseTrigDigFltrTimebaseRate(data)
```

```
SetAnlgWinPauseTrigDigFltrTimebaseSrc(data)
```

```
SetAnlgWinPauseTrigDigSyncEnable(data)
```

```
SetAnlgWinPauseTrigSrc(data)
```

```
SetAnlgWinPauseTrigTop(data)
```

```
SetAnlgWinPauseTrigWhen(data)
```

```
SetAnlgWinRefTrigBtm(data)
```

```
SetAnlgWinRefTrigCoupling(data)
```

```
SetAnlgWinRefTrigDigFltrEnable(data)
```

```
SetAnlgWinRefTrigDigFltrMinPulseWidth(data)
```

```
SetAnlgWinRefTrigDigFltrTimebaseRate(data)
```

```
SetAnlgWinRefTrigDigFltrTimebaseSrc(data)
```

```
SetAnlgWinRefTrigDigSyncEnable(data)
```

```
SetAnlgWinRefTrigSrc(data)
```

```
SetAnlgWinRefTrigTop(data)
```

```
SetAnlgWinRefTrigWhen(data)
```

```
SetAnlgWinStartTrigBtm(data)
```

```
SetAnlgWinStartTrigCoupling(data)
```

```
SetAnlgWinStartTrigDigFltrEnable(data)
```

```
SetAnlgWinStartTrigDigFltrMinPulseWidth(data)
```

```
SetAnlgWinStartTrigDigFltrTimebaseRate(data)
```

```
SetAnlgWinStartTrigDigFltrTimebaseSrc(data)
```

```
SetAnlgWinStartTrigDigSyncEnable(data)
```

```
SetAnlgWinStartTrigSrc(data)
```

```
SetAnlgWinStartTrigTop(data)
```

```
SetAnlgWinStartTrigWhen(data)
```

```
SetArmStartTrigType(data)
```

```
SetBufInputBufSize(data)
```

```
SetBufOutputBufSize(data)
```

```
SetBufOutputOnbrdBufSize(data)
```

```
SetBufferAttribute(attribute)
```

```
SetCIAngEncoderInitialAngle(channel, data)
```

```
SetCIAngEncoderPulsesPerRev(channel, data)
```

```
SetCIAngEncoderUnits(channel, data)
```

```
SetCICountEdgesActiveEdge(channel, data)
```

```
SetCICountEdgesCountDirDigFltrEnable(channel, data)
```

```
SetCICountEdgesCountDirDigFltrMinPulseWidth(channel, data)
```

```
SetCICountEdgesCountDirDigFltrTimebaseRate(channel, data)
```

```
SetCICountEdgesCountDirDigFltrTimebaseSrc(channel, data)
```

```
SetCICountEdgesCountDirDigSyncEnable(channel, data)
```

```
SetCICountEdgesCountResetActiveEdge(channel, data)
```

```
SetCICountEdgesCountResetDigFltrEnable(channel, data)
```

```
SetCICountEdgesCountResetDigFltrMinPulseWidth(channel, data)
```

```
SetCICountEdgesCountResetDigFltrTimebaseRate(channel, data)
```

```
SetCICountEdgesCountResetDigFltrTimebaseSrc(channel, data)
```

```
SetCICountEdgesCountResetDigSyncEnable(channel, data)
```

```
SetCICountEdgesCountResetEnable(channel, data)
```

```
SetCICountEdgesCountResetResetCount(channel, data)
```

```
SetCICountEdgesCountResetTerm(channel, data)
```

```
SetCICountEdgesDigFltrEnable(channel, data)
```

```
SetCICountEdgesDigFltrMinPulseWidth(channel, data)
```

```
SetCICountEdgesDigFltrTimebaseRate(channel, data)
```

```
SetCICountEdgesDigFltrTimebaseSrc(channel, data)
```

```
SetCICountEdgesDigSyncEnable(channel, data)
```

```
SetCICountEdgesDir(channel, data)
```

```
SetCICountEdgesDirTerm(channel, data)
```

```
SetCICountEdgesInitialCnt(channel, data)
```

```
SetCICountEdgesTerm(channel, data)
```

```
SetCICtrTimebaseActiveEdge(channel, data)
```

```
SetCICtrTimebaseDigFltrEnable(channel, data)
```

```
SetCICtrTimebaseDigFltrMinPulseWidth(channel, data)
```

```
SetCICtrTimebaseDigFltrTimebaseRate(channel, data)
```

```
SetCICtrTimebaseDigFltrTimebaseSrc(channel, data)
```

```
SetCICtrTimebaseDigSyncEnable(channel, data)
```

```
SetCICtrTimebaseMasterTimebaseDiv(channel, data)
```

```
SetCICtrTimebaseRate(channel, data)
```

```
SetCICtrTimebaseSrc(channel, data)
```

```
SetCICustomScaleName(channel, data)
```

```
SetCIDataXferMech(channel, data)
```

```
SetCIDataXferReqCond(channel, data)
```

```
SetCIDupCountPrevent(channel, data)
```

```
SetCIEncoderAInputDigFltrEnable(channel, data)
```

```
SetCIEncoderAInputDigFltrMinPulseWidth(channel, data)
```

```
SetCIEncoderAInputDigFltrTimebaseRate(channel, data)
```

```
SetCIEncoderAInputDigFltrTimebaseSrc(channel, data)
```

```
SetCIEncoderAInputDigSyncEnable(channel, data)
```

```
SetCIEncoderAInputTerm(channel, data)
```

```
SetCIEncoderBInputDigFltrEnable(channel, data)
```

```
SetCIEncoderBInputDigFltrMinPulseWidth(channel, data)
```

```
SetCIEncoderBInputDigFltrTimebaseRate(channel, data)
```

```
SetCIEncoderBInputDigFltrTimebaseSrc(channel, data)
```

```
SetCIEncoderBInputDigSyncEnable(channel, data)
```

```
SetCIEncoderBInputTerm(channel, data)
```

```
SetCIEncoderDecodingType(channel, data)
```

```
SetCIEncoderZIndexEnable(channel, data)
```

```
SetCIEncoderZIndexPhase(channel, data)
```

```
SetCIEncoderZIndexVal(channel, data)
```

```
SetCIEncoderZInputDigFltrEnable(channel, data)
```

```
SetCIEncoderZInputDigFltrMinPulseWidth(channel, data)
```

```
SetCIEncoderZInputDigFltrTimebaseRate(channel, data)
```

```
SetCIEncoderZInputDigFltrTimebaseSrc(channel, data)
```

```
SetCIEncoderZInputDigSyncEnable(channel, data)
```

```
SetCIEncoderZInputTerm(channel, data)
```

```
SetCIFreqDigFltrEnable(channel, data)
```

```
SetCIFreqDigFltrMinPulseWidth(channel, data)
```

```
SetCIFreqDigFltrTimebaseRate(channel, data)
```

```
SetCIFreqDigFltrTimebaseSrc(channel, data)
```

```
SetCIFreqDigSyncEnable(channel, data)
```

```
SetCIFreqDiv(channel, data)
```

```
SetCIFreqEnableAveraging(channel, data)
```

```
SetCIFreqMeasMeth(channel, data)
```

```
SetCIFreqMeasTime(channel, data)
```

```
SetCIFreqStartingEdge(channel, data)
```

```
SetCIFreqTerm(channel, data)
```

```
SetCIFreqUnits(channel, data)
```

```
SetCIGPSSyncMethod(channel, data)
```

```
SetCIGPSSyncSrc(channel, data)
```

```
SetCILinEncoderDistPerPulse(channel, data)
```

```
SetCILinEncoderInitialPos(channel, data)
```

```
SetCILinEncoderUnits(channel, data)
```

```
SetCIMax(channel, data)
```

```
SetCIMemMapEnable(channel, data)
```

```
SetCIMin(channel, data)
```

```
SetCIPeriodDigFltrEnable(channel, data)
```

```
SetCIPeriodDigFltrMinPulseWidth(channel, data)
```

```
SetCIPeriodDigFltrTimebaseRate(channel, data)
```

```
SetCIPeriodDigFltrTimebaseSrc(channel, data)
```

```
SetCIPeriodDigSyncEnable(channel, data)
```

```
SetCIPeriodDiv(channel, data)
```

```
SetCIPeriodEnableAveraging(channel, data)
```

```
SetCIPeriodMeasMeth(channel, data)
```

```
SetCIPeriodMeasTime(channel, data)
```

```
SetCIPeriodStartingEdge(channel, data)
```

```
SetCIPeriodTerm(channel, data)
```

```
SetCIPeriodUnits(channel, data)
```

```
SetCIPrescaler(channel, data)
```

```
SetCIPulseFreqDigFltrEnable(channel, data)
```

```
SetCIPulseFreqDigFltrMinPulseWidth(channel, data)
```

```
SetCIPulseFreqDigFltrTimebaseRate(channel, data)
```

```
SetCIPulseFreqDigFltrTimebaseSrc(channel, data)
```

```
SetCIPulseFreqDigSyncEnable(channel, data)
```

```
SetCIPulseFreqStartEdge(channel, data)
```

```
SetCIPulseFreqTerm(channel, data)
```

```
SetCIPulseFreqUnits(channel, data)
```

```
SetCIPulseTicksDigFltrEnable(channel, data)
```

```
SetCIPulseTicksDigFltrMinPulseWidth(channel, data)
```

```
SetCIPulseTicksDigFltrTimebaseRate(channel, data)
```

```
SetCIPulseTicksDigFltrTimebaseSrc(channel, data)
```

```
SetCIPulseTicksDigSyncEnable(channel, data)
```

```
SetCIPulseTicksStartEdge(channel, data)
```

```
SetCIPulseTicksTerm(channel, data)
```

```
SetCIPulseTimeDigFltrEnable(channel, data)
```

```
SetCIPulseTimeDigFltrMinPulseWidth(channel, data)
```

```
SetCIPulseTimeDigFltrTimebaseRate(channel, data)
```

```
SetCIPulseTimeDigFltrTimebaseSrc(channel, data)
```

```
SetCIPulseTimeDigSyncEnable(channel, data)
```

```
SetCIPulseTimeStartEdge(channel, data)
```

```
SetCIPulseTimeTerm(channel, data)
```

```
SetCIPulseTimeUnits(channel, data)
```

```
SetCIPulseWidthDigFltrEnable(channel, data)
```

```
SetCIPulseWidthDigFltrMinPulseWidth(channel, data)
```

```
SetCIPulseWidthDigFltrTimebaseRate(channel, data)
```

```
SetCIPulseWidthDigFltrTimebaseSrc(channel, data)
```

```
SetCIPulseWidthDigSyncEnable(channel, data)
```

```
SetCIPulseWidthStartingEdge(channel, data)
```

```
SetCIPulseWidthTerm(channel, data)
```

```
SetCIPulseWidthUnits(channel, data)
```

```
SetCISemiPeriodDigFltrEnable(channel, data)
```

```
SetCISemiPeriodDigFltrMinPulseWidth(channel, data)
```

```
SetCISemiPeriodDigFltrTimebaseRate(channel, data)
```

```
SetCISemiPeriodDigFltrTimebaseSrc(channel, data)
```

```
SetCISemiPeriodDigSyncEnable(channel, data)
```

```
SetCISemiPeriodStartingEdge(channel, data)
```

```
SetCISemiPeriodTerm(channel, data)
```

```
SetCISemiPeriodUnits(channel, data)
```

```
SetCITimestampInitialSeconds(channel, data)
```

```
SetCITimestampUnits(channel, data)
```

```
SetCITwoEdgeSepFirstDigFltrEnable(channel, data)
```

```
SetCITwoEdgeSepFirstDigFltrMinPulseWidth(channel, data)
```

```
SetCITwoEdgeSepFirstDigFltrTimebaseRate(channel, data)
```

```
SetCITwoEdgeSepFirstDigFltrTimebaseSrc(channel, data)
```

```
SetCITwoEdgeSepFirstDigSyncEnable(channel, data)
```

```
SetCITwoEdgeSepFirstEdge(channel, data)
```

```
SetCITwoEdgeSepFirstTerm(channel, data)
```

```
SetCITwoEdgeSepSecondDigFltrEnable(channel, data)
```

```
SetCITwoEdgeSepSecondDigFltrMinPulseWidth(channel, data)
```

```
SetCITwoEdgeSepSecondDigFltrTimebaseRate(channel, data)
```

```
SetCITwoEdgeSepSecondDigFltrTimebaseSrc(channel, data)
```

```
SetCITwoEdgeSepSecondDigSyncEnable(channel, data)
```

```
SetCITwoEdgeSepSecondEdge(channel, data)
```

```
SetCITwoEdgeSepSecondTerm(channel, data)
```

```
SetCITwoEdgeSepUnits(channel, data)
```

```
SetCIUsbXferReqCount(channel, data)
```

```
SetCIUsbXferReqSize(channel, data)
```

```
SetCOAutoIncrCnt(channel, data)
```

```
SetCOConstrainedGenMode(channel, data)
```

```
SetCOCtrTimebaseActiveEdge(channel, data)
```

```
SetCOCtrTimebaseDigFltrEnable(channel, data)
```

```
SetCOCtrTimebaseDigFltrMinPulseWidth(channel, data)
```

```
SetCOCtrTimebaseDigFltrTimebaseRate(channel, data)
```

```
SetCOCtrTimebaseDigFltrTimebaseSrc(channel, data)
```

```
SetCOCtrTimebaseDigSyncEnable(channel, data)
```

```
SetCOCtrTimebaseMasterTimebaseDiv(channel, data)
```

```
SetCOCtrTimebaseRate(channel, data)
```

```
SetCOCtrTimebaseSrc(channel, data)
```

```
SetCODataXferMech(channel, data)
```

```
SetCODataXferReqCond(channel, data)
```

```
SetCOEnableInitialDelayOnRetrigger(channel, data)
```

```
SetCOMemMapEnable(channel, data)
```

```
SetCOPrescaler(channel, data)
```

```
SetCOPulseDutyCyc(channel, data)
```

```
SetCOPulseFreq(channel, data)
```

```
SetCOPulseFreqInitialDelay(channel, data)
```

```
SetCOPulseFreqUnits(channel, data)
```

```
SetCOPulseHighTicks(channel, data)
```

```
SetCOPulseHighTime(channel, data)
```

```
SetCOPulseIdleState(channel, data)
```

```
SetCOPulseLowTicks(channel, data)
```

```
SetCOPulseLowTime(channel, data)
```

```
SetCOPulseTerm(channel, data)
```

```
SetCOPulseTicksInitialDelay(channel, data)
```

```
SetCOPulseTimeInitialDelay(channel, data)
```

```
SetCOPulseTimeUnits(channel, data)
```

```
SetCOUsbXferReqCount(channel, data)
```

```
SetCOUsbXferReqSize(channel, data)
```

```
SetCOUseOnlyOnBrdMem(channel, data)
```

```
SetChanAttribute(channel, attribute)
```

```
SetChanDescr(channel, data)
```

```
SetChangeDetectDIFallingEdgePhysicalChans(data)
```

```
SetChangeDetectDIRisingEdgePhysicalChans(data)
```

```
SetChangeDetectDITristate(data)
```

```
SetDIAcquireOn(channel, data)
```

```
SetDIDataXferMech(channel, data)
```

```
SetDIDataXferReqCond(channel, data)
```

```
SetDIDigFltrEnable(channel, data)
```

```
SetDIDigFltrEnableBusMode(channel, data)
```

```
SetDIDigFltrMinPulseWidth(channel, data)
```

```
SetDIDigFltrTimebaseRate(channel, data)
```

```
SetDIDigFltrTimebaseSrc(channel, data)
```

```
SetDIDigSyncEnable(channel, data)
```

```
SetDIInvertLines(channel, data)
```

```
SetDILogicFamily(channel, data)
```

```
SetDIMemMapEnable(channel, data)
```

```
SetDITristate(channel, data)
```

```
SetDIUsbXferReqCount(channel, data)
```

```
SetDIUsbXferReqSize(channel, data)
```

```
SetDODataXferMech(channel, data)
```

```
SetDODataXferReqCond(channel, data)
```

```
SetDOGenerateOn(channel, data)
```

```
SetDOInvertLines(channel, data)
```

```
SetDOLineStatesDoneState(channel, data)
```

```
SetDOLineStatesPausedState(channel, data)
```

```
SetDOLineStatesStartState(channel, data)
```

```
SetDOLogicFamily(channel, data)
```

```
SetDOMemMapEnable(channel, data)
```

```
SetDOOutputDriveType(channel, data)
```

```
SetDOOvercurrentAutoReenable(channel, data)
```

```
SetDOOvercurrentLimit(channel, data)
```

```
SetDOOvercurrentReenablePeriod(channel, data)
```

```
SetDOTristate(channel, data)
```

```
SetDOUsbXferReqCount(channel, data)
```

```
SetDOUsbXferReqSize(channel, data)
```

```
SetDOUseOnlyOnBrdMem(channel, data)
```

```
SetDelayFromSampClkDelay(data)
```

```
SetDelayFromSampClkDelayEx(deviceNames, data)
```

```
SetDelayFromSampClkDelayUnits(data)
```

```
SetDelayFromSampClkDelayUnitsEx(deviceNames, data)
```

```
SetDigEdgeAdvTrigDigFltrEnable(data)
```

```
SetDigEdgeAdvTrigEdge(data)
```

```
SetDigEdgeAdvTrigSrc(data)
```

```
SetDigEdgeArmStartTrigDigFltrEnable(data)
```

```
SetDigEdgeArmStartTrigDigFltrMinPulseWidth(data)
```

```
SetDigEdgeArmStartTrigDigFltrTimebaseRate(data)
```

```
SetDigEdgeArmStartTrigDigFltrTimebaseSrc(data)
```

```
SetDigEdgeArmStartTrigDigSyncEnable(data)
```

```
SetDigEdgeArmStartTrigEdge(data)
```

```
SetDigEdgeArmStartTrigSrc(data)
```

```
SetDigEdgeRefTrigDigFltrEnable(data)
```

```
SetDigEdgeRefTrigDigFltrMinPulseWidth(data)
```

```
SetDigEdgeRefTrigDigFltrTimebaseRate(data)
```

```
SetDigEdgeRefTrigDigFltrTimebaseSrc(data)
```

```
SetDigEdgeRefTrigDigSyncEnable(data)
```

```
SetDigEdgeRefTrigEdge(data)
```

```
SetDigEdgeRefTrigSrc(data)
```

```
SetDigEdgeStartTrigDigFltrEnable(data)
```

```
SetDigEdgeStartTrigDigFltrMinPulseWidth(data)
```

```
SetDigEdgeStartTrigDigFltrTimebaseRate(data)
```

```
SetDigEdgeStartTrigDigFltrTimebaseSrc(data)
```

```
SetDigEdgeStartTrigDigSyncEnable(data)
```

```
SetDigEdgeStartTrigEdge(data)
```

```
SetDigEdgeStartTrigSrc(data)
```

```
SetDigEdgeWatchdogExpirTrigEdge(data)
```

```
SetDigEdgeWatchdogExpirTrigSrc(data)
```

```
SetDigLvlPauseTrigDigFltrEnable(data)
```

```
SetDigLvlPauseTrigDigFltrMinPulseWidth(data)
```

```
SetDigLvlPauseTrigDigFltrTimebaseRate(data)
```

```
SetDigLvlPauseTrigDigFltrTimebaseSrc(data)
```

```
SetDigLvlPauseTrigDigSyncEnable(data)
```

```
SetDigLvlPauseTrigSrc(data)
```

```
SetDigLvlPauseTrigWhen(data)
```

```
SetDigPatternPauseTrigPattern(data)
```

```
SetDigPatternPauseTrigSrc(data)
```

```
SetDigPatternPauseTrigWhen(data)
```

```
SetDigPatternRefTrigPattern(data)
```

```
SetDigPatternRefTrigSrc(data)
```

```
SetDigPatternRefTrigWhen(data)
```

```
SetDigPatternStartTrigPattern(data)
```

```
SetDigPatternStartTrigSrc(data)
```

```
SetDigPatternStartTrigWhen(data)
```

```
SetExported10MHzRefClkOutputTerm(data)
```

```
SetExported20MHzTimebaseOutputTerm(data)
```

```
SetExportedAIConvClkOutputTerm(data)
```

```
SetExportedAIHoldCmpltEventOutputTerm(data)
```

```
SetExportedAIHoldCmpltEventPulsePolarity(data)
```

```
SetExportedAdvCmpltEventDelay(data)
```

```
SetExportedAdvCmpltEventOutputTerm(data)
```

```
SetExportedAdvCmpltEventPulsePolarity(data)
```

```
SetExportedAdvCmpltEventPulseWidth(data)
```

```
SetExportedAdvTrigOutputTerm(data)
```

```
SetExportedAdvTrigPulseWidth(data)
```

```
SetExportedAdvTrigPulseWidthUnits(data)
```

```
SetExportedChangeDetectEventOutputTerm(data)
```

```
SetExportedChangeDetectEventPulsePolarity(data)
```

```
SetExportedCtrOutEventOutputBehavior(data)
```

```
SetExportedCtrOutEventOutputTerm(data)
```

```
SetExportedCtrOutEventPulsePolarity(data)
```

```
SetExportedCtrOutEventToggleIdleState(data)
```

```
SetExportedDataActiveEventLvlActiveLvl(data)
```

```
SetExportedDataActiveEventOutputTerm(data)
```

```
SetExportedDividedSampClkTimebaseOutputTerm(data)
```

```
SetExportedHshkEventDelay(data)
```

```
SetExportedHshkEventInterlockedAssertOnStart(data)
```

```
SetExportedHshkEventInterlockedAssertedLvl(data)
```

```
SetExportedHshkEventInterlockedDeassertDelay(data)
```

```
SetExportedHshkEventOutputBehavior(data)
```

```
SetExportedHshkEventOutputTerm(data)
```

```
SetExportedHshkEventPulsePolarity(data)
```

```
SetExportedHshkEventPulseWidth(data)
```

```
SetExportedPauseTrigLvlActiveLvl(data)
```

```
SetExportedPauseTrigOutputTerm(data)
```

```
SetExportedRdyForStartEventLvlActiveLvl(data)
```

```
SetExportedRdyForStartEventOutputTerm(data)
```

```
SetExportedRdyForXferEventDeassertCond(data)
```

```
SetExportedRdyForXferEventDeassertCondCustomThreshold(data)
```

```
SetExportedRdyForXferEventLvlActiveLvl(data)
```

```
SetExportedRdyForXferEventOutputTerm(data)
```

```
SetExportedRefTrigOutputTerm(data)
```

```
SetExportedRefTrigPulsePolarity(data)
```

```
SetExportedSampClkDelayOffset(data)
```

```
SetExportedSampClkOutputBehavior(data)
```

```
SetExportedSampClkOutputTerm(data)
```

```
SetExportedSampClkPulsePolarity(data)
```

```
SetExportedSampClkTimebaseOutputTerm(data)
```

```
SetExportedSignalAttribute(attribute)
```

```
SetExportedStartTrigOutputTerm(data)
```

```
SetExportedStartTrigPulsePolarity(data)
```

```
SetExportedSyncPulseEventOutputTerm(data)
```

```
SetExportedWatchdogExpiredEventOutputTerm(data)
```

```
SetHshkDelayAfterXfer(data)
```

```
SetHshkSampleInputDataWhen(data)
```

```
SetHshkStartCond(data)
```

```
SetHshkTrigType(data)
```

```
SetImplicitUnderflowBehavior(data)
```

```
SetInterlockedHshkTrigAssertedLvl(data)
```

```
SetInterlockedHshkTrigSrc(data)
```

```
SetLoggingFilePath(data)
```

```
SetLoggingFilePreallocationSize(data)
```

```
SetLoggingFileWriteSize(data)
```

```
SetLoggingMode(data)
```

```
SetLoggingPause(data)
```

```
SetLoggingSampsPerFile(data)
```

```
SetLoggingTDMSGroupName(data)
```

```
SetLoggingTDMSOperation(data)
```

```
SetMasterTimebaseRate(data)
```

```
SetMasterTimebaseSrc(data)
```

```
SetOnDemandSimultaneousAOEnable(data)
```

```
SetPauseTrigType(data)
```

```
SetPhysicalChanName(channel, data)
```

```
SetReadAttribute(attribute)
```

```
SetReadAutoStart(data)
```

```
SetReadChannelsToRead(data)
```

```
SetReadOffset(data)
```

```
SetReadOverWrite(data)
```

```
SetReadReadAllAvailSamp(data)
```

```
SetReadRelativeTo(data)
```

```
SetReadSleepTime(data)
```

```
SetReadWaitMode(data)
```

```
SetRealTimeAttribute(attribute)
```

```
SetRealTimeConvLateErrorsToWarnings(data)
```

```
SetRealTimeNumOfWarmupIters(data)
```

```
SetRealTimeReportMissedSamp(data)
```

```
SetRealTimeWaitForNextSampClkWaitMode(data)
```

```
SetRealTimeWriteRecoveryMode(data)
```

```
SetRefClkRate(data)
```

```
SetRefClkSrc(data)
```

```
SetRefTrigAutoTrigEnable(data)
```

```
SetRefTrigDelay(data)
```

```
SetRefTrigPretrigSamples(data)
```

```
SetRefTrigType(data)
```

```
SetSampClkActiveEdge(data)
```

```
SetSampClkDigFltrEnable(data)
```

```
SetSampClkDigFltrMinPulseWidth(data)
```

```
SetSampClkDigFltrTimebaseRate(data)
```

```
SetSampClkDigFltrTimebaseSrc(data)
```

```
SetSampClkDigSyncEnable(data)
```

```
SetSampClkOverrunBehavior(data)
```

```
SetSampClkRate(data)
```

```
SetSampClkSrc(data)
```

```
SetSampClkTimebaseActiveEdge(data)
```

```
SetSampClkTimebaseDiv(data)
```

```
SetSampClkTimebaseMasterTimebaseDiv(data)
```

```
SetSampClkTimebaseRate(data)
```

```
SetSampClkTimebaseSrc(data)
```

```
SetSampClkTimingResponseMode(data)
```

```
SetSampClkUnderflowBehavior(data)
```

```
SetSampQuantSampMode(data)
```

```
SetSampQuantSampPerChan(data)
```

```
SetSampTimingEngine(data)
```

```
SetSampTimingType(data)
```

```
SetStartTrigDelay(data)
```

```
SetStartTrigDelayUnits(data)
```

```
SetStartTrigRetriggerable(data)
```

```
SetStartTrigType(data)
```

```
SetSwitchScanAttribute(attribute)
```

```
SetSwitchScanBreakMode(data)
```

```
SetSwitchScanRepeatMode(data)
```

```
SetSyncClkInterval(data)
```

```
SetSyncPulseMinDelayToStart(data)
```

```
SetSyncPulseResetDelay(data)
```

```
SetSyncPulseSrc(data)
```

```
SetTimingAttribute(attribute)
```

```
SetTimingAttributeEx(deviceNames, attribute)
```

```
SetTrigAttribute(attribute)
```

```
SetTriggerSyncType(data)
```

```
SetWatchdogAOExpirState(lines, data)
```

```
SetWatchdogAOOutputType(lines, data)
```

```
SetWatchdogAttribute(lines, attribute)
```

```
SetWatchdogCOExpirState(lines, data)
```

```
SetWatchdogDOExpirState(lines, data)
```

```
SetWatchdogExpirTrigTrigOnNetworkConnLoss(data)
```

```
SetWatchdogExpirTrigType(data)
```

```
SetWatchdogTimeout(data)
```

```
SetWriteAttribute(attribute)
```

```
SetWriteNextWriteIsLast(data)
```

```
SetWriteOffset(data)
```

```
SetWriteRegenMode(data)
```

```
SetWriteRelativeTo(data)
```

```
SetWriteSleepTime(data)
```

```
SetWriteWaitMode(data)
```

```
StartNewFile(filePath)
```

```
StartTask()
```

```
StopTask()
```

```
TaskControl(action)
```

```
WaitForNextSampleClock(timeout, isLate)
```

```
WaitUntilTaskDone(timeToWait)
```

```
WriteAnalogF64(numSampsPerChan, autoStart, timeout, dataLayout, writeArray,
sampsPerChanWritten, reserved)
```

```
WriteAnalogScalarF64(autoStart, timeout, value, reserved)
```

```
WriteBinaryI16(numSampsPerChan, autoStart, timeout, dataLayout, writeArray,
sampsPerChanWritten, reserved)
```

```
WriteBinaryI32(numSampsPerChan, autoStart, timeout, dataLayout, writeArray,
sampsPerChanWritten, reserved)
```

```
WriteBinaryU16(numSampsPerChan, autoStart, timeout, dataLayout, writeArray,
sampsPerChanWritten, reserved)
```

```
WriteBinaryU32(numSampsPerChan, autoStart, timeout, dataLayout, writeArray,
sampsPerChanWritten, reserved)
```

```
WriteCtrFreq(numSampsPerChan, autoStart, timeout, dataLayout, frequency, dutyCycle,
numSampsPerChanWritten, reserved)
```

```
WriteCtrFreqScalar(autoStart, timeout, frequency, dutyCycle, reserved)
```

```
WriteCtrTicks(numSampsPerChan, autoStart, timeout, dataLayout, highTicks, lowTicks,
numSampsPerChanWritten, reserved)
```

```
WriteCtrTicksScalar(autoStart, timeout, highTicks, lowTicks, reserved)
```

```
WriteCtrTime(numSampsPerChan, autoStart, timeout, dataLayout, highTime, lowTime,
numSampsPerChanWritten, reserved)
```

```
WriteCtrTimeScalar(autoStart, timeout, highTime, lowTime, reserved)
```

```
WriteDigitalLines(numSampsPerChan, autoStart, timeout, dataLayout, writeArray,
sampsPerChanWritten, reserved)
```

```
WriteDigitalScalarU32(autoStart, timeout, value, reserved)
```

```
WriteDigitalU16(numSampsPerChan, autoStart, timeout, dataLayout, writeArray,
sampsPerChanWritten, reserved)
```

```
WriteDigitalU32(numSampsPerChan, autoStart, timeout, dataLayout, writeArray,
sampsPerChanWritten, reserved)
```

```
WriteDigitalU8(numSampsPerChan, autoStart, timeout, dataLayout, writeArray,
sampsPerChanWritten, reserved)
```

```
WriteRaw(numSamps, autoStart, timeout, writeArray, sampsPerChanWritten, reserved)
```

```
start()
```

```
stop()
```

Core

DAQmx is a python wrapper to the National Instruments DAQmx driver. You can use *core* to access the driver in c-style.

National Instruments does not allow to copy any documentation so you have to search C-api reference for the method name.

You need the `io` feature in your license to unlock the DAQmx features.

Changelog

htf-daqmx-3.2.0

- ship `cp312-abi3` wheels that are compatible with Python 3.12 and newer versions

htf-daqmx-3.1.6

- implement Cyber Resilience Act Conformity (CRA)
- update dependencies

htf-daqmx-3.1.5

- add support for Python 3.13

htf-daqmx-3.1.4

- add requirements for validation purposes

htf-daqmx-3.1.3

- add build environment data for validation purposes

htf-daqmx-3.1.2

- `htf-daqmx` can now be used standalone

htf-daqmx-3.1.1

- add support for Python 3.12

htf-daqmx-3.1.0

- add type-hints
- use `hlm-3.1`

htf-daqmx-3.0.1

- add support for Python 3.11

htf-daqmx-3.0.0

- extract `htf-daqmx` from `htf`

Power Supplies

htf support some programmable power supplies.

Rohde + Schwarz HMP Series programmable power supplies

The Rhode + Schwarz HMP Series are programmable power supplies that are supported by the HILSTER Testing Framework.

The user does not need not call any methods. He can use the automatic delegation to easily set current and voltage by assigning values to the members of the same names.

The abstraction supports virtual controls that appear as class members. These are `voltage<i>`, `actual_voltage<i>`, `current<i>` and `actual_current<i>`. `<i>` it the port `(1..4)`.

For most usecases using the virtual controls should be appropriate.

To setup a HMP 2020 power supply use:

```
from htf.power supply import HMP2020

h = HMP2020("COM3")

# set the voltage on port 1
h.voltage1 = 1.23
# print the measured voltage on port 1
print(h.actual_voltage1)

# set the current on port 2
h.current2 = 0.2
# print the measured current on port 2
print(h.actual_current2)
```

To make use of R&S®HO720 or R&S®HO730 Dual Interface via ethernet you can simply pass a url e.g.

```
h = HMP4040("socket://hmp4040-ip-address:5025")  # change to your ip address or hostname
```

```
class htf.powersupply.rs_hmpseries.HMP2020(port: str, baudrate: int = 9600,
parity: str = 'N', stopbits: int | float = 1, timeout: int | float = 1.0, command_delay: int
| float = 0.05, wait_after_set_voltage: bool = False, wait_time: int | float = 0.5)
```

HMP2020: 1 × 0 V to 32 V/0 A to 10 A; 1 × 0 V to 32 V/0 A to 5 A (188 W)

```
class htf.powersupply.rs_hmpseries.HMP2030(port: str, baudrate: int = 9600,
parity: str = 'N', stopbits: int | float = 1, timeout: int | float = 1.0, command_delay: int
| float = 0.05, wait_after_set_voltage: bool = False, wait_time: int | float = 0.5)
```

HMP2030: 3 × 0 V to 32 V/0 A to 5 A (188 W)

```
class htf.powersupply.rs_hmpseries.HMP4030(port: str, baudrate: int = 9600,
parity: str = 'N', stopbits: int | float = 1, timeout: int | float = 1.0, command_delay: int
| float = 0.05, wait_after_set_voltage: bool = False, wait_time: int | float = 0.5)
```

HMP4030: 3 × 0 V to 32 V/0 A to 10 A (384 W)

```
class htf.powersupply.rs_hmpseries.HMP4040(port: str, baudrate: int = 9600,
parity: str = 'N', stopbits: int | float = 1, timeout: int | float = 1.0, command_delay: int
| float = 0.05, wait_after_set_voltage: bool = False, wait_time: int | float = 0.5)
```

HMP4040: 4 × 0 V to 32 V/0 A to 10 A (384 W)

Manson HCS-3XXX

The Manson HCS-3XXX is a programmable power supply unit family.

The user does not need not call any methods. He can use the automatic delegation to easily set current and voltage by assigning values to the members of the same names.

```
from htf.powersupply import MansonHCS3xxx

m - MansonHCS3xxx("COM3")

# set output to 4 V
m.voltage = 4
print(m.voltage)
# set current (maximum) to 0.5 A
m.current = 0.5
print(m.current)
```

To store presets for current and voltage levels use the `store_presets` method. You can store up to three presets.

```
presets = {
    'P1': {'current': 12.0, 'voltage': 5.0},
    'P2': {'current': 12.0, 'voltage': 13.8},
    'P3': {'current': 12.0, 'voltage': 25.0}}
m.store_presets(presets)
```

```
class htf.powersupply.manson_hcs_3xxx.MansonHCS3xxx(port: str, baudrate: int = 9600, parity:
str = 'N', stopbits: int | float = 1, timeout: int | float = 1.0, decimal_places_for_current:
int = 1, debug: bool = False)
```

`MansonHCS3xxx` is a hardware abstraction to the Manson HCS-3xxx power supply family using a UART to control it.

`MansonHCS3xxx` uses `DelegatorMixin` to automatically set and read it's members *voltage* and *current*.

Parameters:

- **port** – the comport to be used, eg. "COM3", 3 or "/dev/ttyUSB0", etc.
- **decimal_places_for_current=1** – the number of decimal places for the current. HCS-3100, 3150, 3200 and 3202 have one decimal place. HCS-3102, 3014 and 3204 have two decimal places.
- **debug=False** – if set to *True* debugging is enabled

```
close() → None
```

Close the serial member. This method is automatically called when `__del__` is called.

```
disable_output() → None
```

Disable output.

```
enable_output() → None
```

Enable output.

```
get_current() → float
```

Read the current current from device and return the result.

Returns:
the current current.
Return type:
float

```
get_display_voltage_current_and_status() → Dict[str, Any]
```

Get the display voltage, current and status.

Returns:
a dictionary containing “voltage”, “current” and “status”.
Return type:
dict

```
get_preset_maximum_current() → float
```

Get preset maximum current for current preset.

Returns:
the preset maximum current.
Return type:
float

```
get_preset_maximum_voltage() → float
```

Get preset maximum voltage for current preset.

Returns:
the preset maximum voltage.
Return type:
float

```
get_preset_maximum_voltage_and_current() → Dict[str, float]
```

Get the preset maximum voltage and current.

Returns:
a dictionary containing “voltage” and “current”.
Return type:
dict

```
get_preset_voltage_and_current() → Dict[str, float]
```

Get the preset voltage and current.

Returns:
a dictionary containing “voltage” and “current”.
Return type:
dict

```
get_presets() → Dict[str, Dict[str, float]]
```

Get three voltage-current presets.

Returns:
a dictionary containing “voltage”, “current” and “P<preset>”.
Return type:
dict

```
get_voltage() → float
```

Read current voltage from device and return the result.

Returns:
the current voltage.
Return type:
float

```
load_preset(preset: int) → None
```

Load preset `preset`.

Parameters:
preset – may be 0, 1, or 2.

```
preset_maximum_current(current: int | float) → None
```

Set maximum voltage for current preset.

```
preset_maximum_voltage(voltage: int | float) → None
```

Set maximum voltage for current preset.

```
query(query: bytes, answer_lines: int = 1) → List[bytes] | None
```

Send a command and wait for the answer.

Parameters:

- **query** – the query to be sent.
- **answer_lines=1** – the number of lines to be received.

Returns:
the answer line(s).
Return type:
str
Raises:
manson_hcs_3xxx.TimeoutException – if a communication time-out occurred.

```
set_current(current: int | float) → None
```

Set the output current to `current`.

Parameters:
current – the current in units of amperes as int(), float() or string, eg. 0, 0.5, “0.5” or even “0.5 mA” work.

```
set_voltage(voltage: int | float) → None
```

Set the output voltage to `voltage`.

Parameters:
voltage – the voltage in units of volts as int(), float() or string, eg. 23, 23.0, “23.0” or even “23.0 V” work. *voltage* may have on decimal point. If *voltage* is `None` the output is disabled.

```
store_presets(presets: Dict[str, Dict[str, float]]) → None
```

Store three voltage-current presets.

Parameters:
presets – a dictionary containing the key P1, P2 and P3. Each key contains “voltage” and “current” keys.

KORAD KA3005P

The KORAD KA3005P is a programmable power supply unit.

To use the implementation you do not need to call any methods. Simply use the automatic delegation to easily set current and voltages by assigning values to the members of the same name.

```
from htf.powersupply import KORAD_KA3005P

k = KORAD_KA3005P("COM3")

# set output to 4 V
k.voltage = 4
print(k.voltage)
# set current (maximum) to 0.5 A
k.current = 0.5
print(k.current)
```

```
class htf.powersupply.korad_ka3005p.KORAD_KA3005P(port: str, baudrate: int = 9600, parity:
str = 'N', stopbits: int | float = 1, timeout: int | float = 1.0, debug: bool = False)
```

`KORAD_KA3005P` is a hardware abstraction to the KORAD KA3005P power supply unit using a UART to control it.

`KORAD_KA3005P` uses `DelegatorMixin` to automatically set and read it's members `voltage` and `current`.

Parameters:

- **port** – the comport to be used, eg. "COM3", 3 or "/dev/ttyUSB0", etc.
- **debug=False** – if set to `True` debugging is enabled.

```
close() → None
```

Close the serial member. This method is automatically called when `__del__` is called.

```
disable_output() → None
```

Disable output.

```
enable_output() → None
```

Enable output.

Raises:
SetOutputException – if output was not set.

```
get_current_current(channel: int = 1) → float
```

Read the current current from device and return the result.

Parameters:
channel – the channel to use. Shall always be 1.
Returns:
the current current.
Return type:
float

```
get_current_voltage(channel: int = 1) → float
```

Read current voltage from device and return the result.

Parameters:
channel=1 – the channel to use. Shall always be 1.
Returns:
the current voltage.
Return type:
float

```
get_id() → bytes | None
```

Return the KORAD id.

Returns:
the KORAD id.
Return type:
str

```
get_status() → Dict[str, str]
```

Read device status and return a dictionary.

Returns:
the status dictionary containing the keys “ch1 mode”, “ch2 mode”, “tracking mode”, “beep”,
”lock” and “output”.

Return type:
dict

```
query(query: bytes, length_of_answer: int | None = None, retries: int = 3) → bytes
```

Send a command and wait for the answer.

Parameters:

- **query** – the query to be sent.
- **length_of_answer=None** – the maximum number of bytes to be received.

- **retries=3** – the number of retries.

Returns:
the answer.
Return type:
bytes
Raises:
korad_ka3005p.TimeoutException – if a communication time-out occurred.

```
send_command(command: bytes) → None
```

Directly send a command to the power supply unit.

Parameters:
command – the command to be sent.

```
set_current(current: int | float, channel: int = 1) → None
```

Set the output current for a given channel.

Parameters:

- **current** – the current in units of amperes as int(), float() or string, eg. 0, 0.5, “0.5” or even “0.5 mA” work.
- **channel** – the channel to use. Shall always be 1.

Raises:
SetCurrentException – if current was not set.

```
set_voltage(voltage: int | float, channel: int = 1) → None
```

Set the output voltage for a given channel.

If `voltage` is None the output is disabled. If `voltage` is not None the output is enabled. If `voltage` changes the output is first disabled, changed, and then enabled again.

Parameters:

- **voltage** – the voltage in units of volts as int(), float() or string, eg. 23, 23.0, “23.0”
- **work.** (*or even "23.0 V"*)
- **channel** – the channel to use. Shall always be 1.

Raises:
SetVoltageException – if voltage was not set.

Changelog

htf-powersupply-4.1.0

- ship `cp312-abi3` wheels that are compatible with Python 3.12 and newer versions

htf-powersupply-4.0.6

- implement Cyber Resilience Act Conformity (CRA)
- update dependencies

htf-powersupply-4.0.5

- add support for Python 3.13

htf-powersupply-4.0.4

- add requirements for validation purposes

htf-powersupply-4.0.3

- add build environment data for validation purposes

htf-powersupply-4.0.2

- `htf-powersupply` can now be used standalone

htf-powersupply-4.0.1

- add support for Python 3.12

htf-powersupply-4.0.0

- remove all camel-case methods
- add type-hints
- use `hlm-3.1`
- rename `comport` to `port` for all HMP power supplies

htf-powersupply-3.0.1

- add support for Python 3.11

htf-powersupply-3.0.0

- extract `htf-powersupply` from `htf`

Retry Functions

The retry functions can be used to do automatic retries. There are three different ways to use them. `htf.retry` can be called directly. `htf.retriable` is a decorator that can be used to decorate functions so that they are retried and `htf.retrying` is a context manager that can be used to do retries within a context block.

If an exception occurs during execution of the action, it is retried until the maximum number of attempts is reached. Exceptions during the last retry are re-raised to inform the caller of its occurence.

The delay between attempts can also be specified. By default there is no delay.

Direct Calls

Direct calls are done using `htf.retry` or `htf.retry_sync`.

Using coroutines you can run:

```
import htf

count = 0

async def foo(bar):
    global count

    print("foo() called with", bar)
    print("count:", count)

    count += 1

    if count < 3:
        raise ValueError("count is too small")

    return "success"


await htf.retry(foo, args=("bar",), attempts=3, delay=1.0)
```

To avoid using coroutines, you can run:

```
import htf

count = 0

def foo(bar):
    global count

    print("foo() called with", bar)
    print("count:", count)

    count += 1

    if count < 3:
        raise ValueError("count is too small")

    return "success"


htf.retry_sync(foo, args=("bar",), attempts=3, delay=1.0)
```

```
async htf.retry(action: ~typing.Callable[[...], ~typing.Any], args: ~typing.Tuple | None =
None, kwargs: ~typing.Dict[str, ~typing.Any] | None = None, attempts: int = 2, delay: float |
None = None, retry_exceptions: ~typing.Tuple[~typing.Any] = (<class 'BaseException'>,)) → Any
```

Retry `action` with a given number of attempts and a given delay.

Parameters:

- **action** – the action to be retried (coroutines are supported)
- **args=()** – the positional arguments passed to `action`
- **kwargs={}** – the keyword arguments passed to `action`
- **attempts=2** – the number of attempts
- **delay=None** – the delay between attempts in seconds
- **retry_exceptions=** (*BaseException,*) – the exceptions to be catched to retry the action

Returns:
the return value of `action(*args, **kwargs)`

```
htf.retry_sync(action: ~typing.Callable[[...], ~typing.Any], args: ~typing.Tuple | None =
None, kwargs: ~typing.Dict[str, ~typing.Any] | None = None, attempts: int = 2, delay: float |
None = None, retry_exceptions: ~typing.Tuple[~typing.Any] = (<class 'BaseException'>,)) → Any
```

Retry `action` with a given number of attempts and a given delay.

Parameters:

- **action** – the action to be retried (coroutines are not supported)
- **args=()** – the positional arguments passed to `action`
- **kwargs={}** – the keyword arguments passed to `action`
- **attempts=2** – the number of attempts
- **delay=None** – the delay between attempts in seconds
- **retry_exceptions=** (*BaseException,*) – the exceptions to be catched to retry the action

Returns:
the return value of `action(*args, **kwargs)`

Using the Decorator

If a function or method is supposed to always do retries, the decorator `htf.retriable` can be used. If the decorated method is a coroutine the result is a coroutine, too.

```python
import htf

count = 0

@htf.retriable(attempts=3, delay=1.0)
def foo(bar):
    global count

    print("foo() called with", bar)
```

```
    print("count:", count)

    count += 1

    if count < 3:
        raise ValueError("count is too small")

    return "success"


foo("bar")
```

```
htf.retriable(attempts: int = 2, delay: float | None = None, retry_exceptions: Tuple[Any] =
(<class 'BaseException'>, )) → Callable[[Callable[[...], Any]], Any]
```

A decorator factory for retry(). Wrap your function in @retriable(...) to give it retry powers!

Parameters:

- **attempts=2** – the number of attempts
- **delay=None** – the delay between attempts in seconds
- **retry_exceptions=** (*BaseException,*) – the exceptions to be catched to retry the action

Returns:

a function decorator supporting retries. If a coroutine is supplied a coroutine is returned

else not.

Using the Context Manager

To use retries within a context block use the `htf.retrying` context manager.

```
import htf

count = 0

def foo(bar):
    global count

    print("foo() called with", bar)
    print("count:", count)

    count += 1

    if count < 3:
        raise ValueError("count is too small")

    return "success"


with htf.retrying(foo, attempts=3, delay=1.0) as retrying_foo:
    await retrying_foo("bar")
```

```
htf.retrying(action: ~typing.Callable[[...], ~typing.Any], attempts: int = 2, delay: float |
None = None, retry_exceptions: ~typing.Tuple[~typing.Any] = (<class 'BaseException'>,)) →
ContextManager | AsyncContextManager
```

A context manager for wrapping `action` with retry functionality with a given number of attempts and a given delay.

Parameters:

- **action** – the action to be retried
- **attempts=2** – the number of attempts
- **delay=None** – the delay between attempts in seconds
- **retry_exceptions=** (*BaseException,*) – the exceptions to be catched to retry the action

Returns:
a context manager supporting retries. If a coroutine is supplied an async context manager
is returned.

Issue Tracking

The HILSTER Testing Framework seamlessly integrates issue tracking systems.

JIRA (Atlassian)

The integration of JIRA issues can easily be realized using the `jira` fixture.

In JIRA the option `'Allow Remote API Calls'` must be enabled.

The `jira` fixture searches JIRA using the rest api to find issues for the current test. Found issues are appended to the test and put into the test report.

This way you do not need to put your issues into the source code of your tests.

By default issues must include `testcase.test` in the summary or the description, eg. `MyTestCase.test_foo`. This behaviour can be changed by changing the `query_template` accordingly.

To use the `jira` fixture you need to create your own fixture depending on it to provide the necessary JIRA URL and authentication credentials. The fixture should have the scope “session” and have the `auto` parameter set to `True`.

By default the `jira` fixture creates links that include the type, id and status of the referenced issue, e.g.
`Bug TRANS-1266 (Resolved): Dutch language does not reflect on keyboard shortcut dialog box`.

The default template for `link_template` is `"{fields/issuetype/name} {key} ({fields/status/name}): {fields/summary}"`. The `'/'` in the template describe the hierarchy in the api result. The api result is flattened and is used to format the template using the `string.format` method.

To be able to see all possible keys you should enable debugging to have a look at the flattend dictionary.

For `oauth` and `kerberos` authentication have a look at the python-jira documentation.

Example:

```
import htf

@htf.fixture('session', auto=True)
def setup_jira(jira):
    jira.setup_jira_connector(
        url="https://jira.atlassian.com",
        project="11460"
    )

def test_foo():
    pass
```

To enable debugging use the following code snippet.

```
import logging
logging.basicConfig()
logger = logging.getLogger("htf.jira")
logger.setLevel(level=logging.INFO)  # to enable info
logger.setLevel(level=logging.DEBUG)  # to enable info and debug
```

```
class htf.fixtures.jira(environment: TestEnvironment)
```

```
setup_jira_connector(url: str, project: str, username: str | None = None, password: str
| None = None, auth: str | None = None, oauth_dict: Dict[Any, Any] | None = None,
kerberos_options: Dict[Any, Any] | None = None, query_template: str | None = None,
title_template: str | None = None) → None
```

Set up the `jira` fixture so that it will search your JIRA installation for issues for the current test.

Parameters:

- **url** – the url to your JIRA installation
- **project** – the JIRA project
- **username=None** – the username to be used for authentication
- **password=None** – the password to be used for authentication
- **auth=None** –
  possible values are:
  - `None`: no authentication is used
  - `auth`: cookie based authentication

    - `basic_auth`: HTTP basic authentication
    - `oauth`: OAuth authentication is used (you have to supply `oauth_dict`, too)
    - `kerberos`: Kerberos authentication (you have to supply `kerberos_options`, too)
- **oauth_dict=None** – options for `oauth` authentication
- **kerberos_options=None** – options for `kerberos` authentication
- **query_template="default"** –
  the template for the JIRA query string. supported replacements are - `{testcase}`: is replaced with the TestCase's class name - `{test}`: is replaced with the current test name - `{project}`: is replaced with the supplied project
  default: "project={project} and (summary~"{testcase}.{test}" or description~"{testcase}.{test}")"
- **title_template="default"** – The template for the resulting title of the link in the test report. The issue is flattened using `'/'` as the separator. So the template may include keys like `{fields/status/name}` to show the name of the current ticket status, etc.

Thread — A traceable thread implementation

```
class htf.Thread(group: None = None, target: Callable[[...], Any] | None = None, name: str |
None = None, stop: Callable[[...], bool] | None = None, force_stop: bool = True, args:
Tuple[Any] | None = None, kwargs: Dict[str, Any] | None = None)
```

`Thread` extends `threading.Thread` to have a possibility to traceback where it was started.

This constructor should always be called with keyword arguments. Arguments are:

group should be None; reserved for future extension when a ThreadGroup class is implemented.

target is the callable object to be invoked by the run() method. Defaults to None, meaning nothing is called.

name is the thread name. By default, a unique name is constructed of the form "Thread-N" where N is a small decimal number.

args is the argument tuple for the target invocation. Defaults to ().

kwargs is a dictionary of keyword arguments for the target invocation. Defaults to {}.

If a subclass overrides the constructor, it must make sure to invoke the base class constructor (Thread.__init__()) before doing anything else to the thread.

```
is_alive() → bool
```

Return whether the thread is alive.

This method returns True just before the run() method starts until just after the run() method terminates. The module function enumerate() returns a list of all alive threads.

```
join(timeout: float | None = None) → None
```

Wait until the thread terminates.

This blocks the calling thread until the thread whose join() method is called terminates – either normally or through an unhandled exception or until the optional timeout occurs.

When the timeout argument is present and not None, it should be a floating point number specifying a timeout for the operation in seconds (or fractions thereof). As join() always returns None, you must call is_alive() after join() to decide whether a timeout happened – if the thread is still alive, the join() call timed out.

When the timeout argument is not present or None, the operation will block until the thread terminates.

A thread can be join()ed many times.

join() raises a RuntimeError if an attempt is made to join the current thread as that would cause a deadlock. It is also an error to join() a thread before it has been started and attempts to do so raises the same exception.

```
run() → None
```

Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

```
start() → None
```

Start the thread's activity.

It must be called at most once per thread object. It arranges for the object's run() method to be invoked in a separate thread of control.

This method will raise a RuntimeError if called more than once on the same thread object.

Usage:

```
>>> def _thread():
...    print("test")

>>> thread = Thread(name="my-thread", target=_thread)
>>> thread.start()
>>> print(thread)
... <Thread(my-thread, started 1337,
...   File "/path/to/thread.py", line 23, in <module>
```

```
...     thread.start()
... )>
```

Metadata

Tests can have metadata, i.e. the source filename, revision, id, etc. .

```
class htf.MetaData(value: str | bytes)
```

`MetaData` can be used to mark variables to be handled as meta data.

For example `MetaData` can be used to set additional attributes of a test case that is put into the meta data section of a test report.

Initialize self. See help(type(self)) for accurate signature.

Example usage:

```
import htf


class ExampleTestCase(htf.TestCase):
    Id = htf.MetaData("$Id: example.py, author, deadbeef $")
    Revision = htf.MetaData("$Revision: deadbeef $")

    def test_1(self):
        pass

if __name__ == "__main__":
    htf.main()
```

Keywords

Commonly keywords are used in source code, eg. `$Id: ...$`.

There are several metadata aliases for these keywords. The resulting metadata the content but not the keywords that is actually expanded.

```
class htf.Id(value: str | bytes)
```

Returns “an id” in test report.

Initialize self. See help(type(self)) for accurate signature.

Example Usage:

```
Id("$Id$")
```

```
class htf.Revision(value: str | bytes)
```

Returns “a revision” in test report.

Initialize self. See help(type(self)) for accurate signature.

Example Usage:

```
class Test(TestCase):

    Revision = Revision("$Revision$")
```

```
class htf.Rev(value: str | bytes)
```

Returns “a revision” in test report.

Alias of `Revision`.

Initialize self. See help(type(self)) for accurate signature.

Example Usage:

```
class Test(TestCase):

    Rev = Rev("$Rev$")
```

```
class htf.LastChangedRevision(value: str | bytes)
```

Returns “a revision” in test report.

Alias of `Revision`.

Initialize self. See help(type(self)) for accurate signature.

Example Usage:

```
class Test(TestCase):

    LastChangedRevision = LastChangedRevision("$LastChangedRevision$")
```

```
class htf.Date(value: str | bytes)
```

Returns “a date” in test report.

Initialize self. See help(type(self)) for accurate signature.

Example Usage:

```
class Test(TestCase):

    Date = Date("$Date$")
```

```
class htf.LastChangedDate(value: str | bytes)
```

Returns “a date” in test report.

Alias of `Date`.

Initialize self. See help(type(self)) for accurate signature.

Example Usage:

```
class Test(TestCase):

    LastChangedDate = LastChangedDate("$LastChangedDate$")
```

```
class htf.Header(value: str | bytes)
```

Returns “a header” in test report.

Initialize self. See help(type(self)) for accurate signature.

Example Usage:

```
class Test(TestCase):

    Header = Header("$Header$")
```

```
class htf.Author(value: str | bytes)
```

Returns “an author” in test report.

Initialize self. See help(type(self)) for accurate signature.

Example Usage:

```
class Test(TestCase):

    Author = Author("$Author$")
```

```
class htf.LastChangedBy(value: str | bytes)
```

Returns the author of the last modification of the document.

Alias of `Author`.

Initialize self. See help(type(self)) for accurate signature.

Example Usage:

```
class Test(TestCase):

    LastChangedBy = LastChangedBy("$LastChangedBy$")
```

```
class htf.Source(value: str | bytes)
```

Returns: “a source file” in test report.

Initialize self. See help(type(self)) for accurate signature.

Example Usage:

```
class Test(TestCase):

    Source = Source("$Source$")
```

```
class htf.HeadURL(value: str | bytes)
```

Returns: “https://server:port:/file” in test report.

Initialize self. See help(type(self)) for accurate signature.

Example Usage:

```
class Test(TestCase):

    HeadURL = HeadURL("$HeadURL$")
```

```
class htf.URL(value: str | bytes)
```

Returns: “https://server:port:/file” in test report.

Alias of `HeadURL`.

Initialize self. See help(type(self)) for accurate signature.

Example Usage:

```
class Test(TestCase):

    URL = URL("$URL$")
```

```
class htf.RCSFile(value: str | bytes)
```

Returns: RCSFile marker

Initialize self. See help(type(self)) for accurate signature.

Example Usage:

```
class Test(TestCase):

    RCSFile = RCSFile("$RCSFile$")
```

```
class htf.RCSfile(value: str | bytes)
```

Returns: RCSfile marker

Alias of `RCSFile`.

Initialize self. See help(type(self)) for accurate signature.

Example Usage:

```
class Test(TestCase):

    RCSfile = RCSfile("$RCSfile$")
```

Miscellaneous

```
class htf.markers.TestingframeworkVersion
```

Returns: “htf-<htf-version> on Python <python-version>” in test report.

Initialize self. See help(type(self)) for accurate signature.

Example Usage:

```
class Test(TestCase):

    testingframeworkVersion = TestingframeworkVersion()
```

Filters — Building generic filter expressions

`filters` is a generic filter expression module to build arbitrary callable filters. A filter expression returns `True` or `False`.

NOT Filter

Functions as an invert of an individual filter.

```
filter_not(filter)
```

AND Filter

Returns `True` if all filter expressions return `True` individually.

```
filter_and(filter1, filter2, ..., filterN)
```

NAND Filter

Functions as an inverted AND Filter (`filter_and()`).

```
filter_nand(filter1, filter2, ..., filterN)
```

OR Filter

Returns `True` as soon as any of the filter expressions returns `True` individually.

```
filter_or(filter_or(filter1, filter2, ..., filterN))
```

NOR Filter

Functions as an inverted OR Filter (`filter_or()`).

```
filter_nor(filter1, filter2, ..., filterN)
```

XOR Filter

Returns `True` if an odd number of filters individually returns `True`. In this, Exceptions are rated as a filter returning `False`.

```
filter_xor(filter1, filter2, ..., filterN)
```

XNOR Filter

Functions as an inverted XOR Filter (`filter_xor()`).

```
filter_xnor(filter1, filter2, ..., filterN)
```

Examples

A possible example for an OR-Filter is:

```
from htf.filters import filter_or

def filter1(arg):
    return arg == 1

def filter2(arg):
    return arg == 2

filter1(1)  # True
filter1(2)  # False

filter1or2 = filter_or(filter1, filter2)  # create a new callable filter that matches 1 or 2

filter1or2(1)  # True
```

```
filter1or2(2)  # True
filter1or2(3)  # False
```

A possible example for a XOR-Filter is:

```
from htf.filters import filter_xor

t = lambda *args, **kwargs: True

print(filter_xor(t)())  # True
print(filter_xor(t, t)())  # False
print(filter_xor(t, t, t)())  # True
print(filter_xor(True, False)())  # True
```

Filters

```
htf.filters.filter_and(*filters: Callable[[...], bool]) → Callable[[...], bool]
```

Create a filter that combines all results or values of `*filters` using logical and.

Parameters:
***filters** – a tuple of filters.
Returns:
a filter that combines all elements in `*filters` using logical and.
Return type:
callable

```
htf.filters.filter_nand(*filters: Callable[[...], bool]) → Callable[[...], bool]
```

Create a filter that combines all results or values of `*filters` using logical nand.

Parameters:
***filters** – a tuple of filters.
Returns:
a filter that combines all elements in `*filters` using logical nand.
Return type:
callable

```
htf.filters.filter_nor(*filters: Callable[[...], bool]) → Callable[[...], bool]
```

Create a filter that combines all results or values of `*filters` using logical nor.

Parameters:
***filters** – a tuple of filters.
Returns:
a filter that combines all elements in `*filters` using logical nor.

Return type:
callable

```
htf.filters.filter_not(filter_method: Callable[[...], bool]) → Callable[[...], bool]
```

Create a filter that inverts `filter_method`'s result or values.

Parameters:
filter_method (*callable*) – the filter that is inverted.
Returns:
a filter that inverts `filter_method`.
Return type:
callable

```
htf.filters.filter_or(*filters: Callable[[...], bool]) → Callable[[...], bool]
```

Create a filter that combines all results or values of `*filters` using or-logic.

Parameters:
***filters** – a tuple of filters.
Returns:
a filter that conjuncts all elements in `*filters`.
Return type:
callable

```
htf.filters.filter_xnor(*filters: Callable[[...], bool]) → Callable[[...], bool]
```

Create a filter that combines all results or values of `*filters` using logical xnor.

Parameters:
***filters** – a tuple of filters.
Returns:
a filter that combines all elements in `*filters` using logical xnor.
Return type:
callable

```
htf.filters.filter_xor(*filters: Callable[[...], bool]) → Callable[[...], bool]
```

Create a filter that combines all results or values of `*filters` using xor-logic.

Parameters:
***filters** – a tuple of filters.
Returns:
a filter that xors all elements in `*filters`.
Return type:
callable

Changelog

htf-filters-4.1.0

- ship `cp312-abi3` wheels that are compatible with Python 3.12 and newer versions

htf-filters-4.0.6

- implement Cyber Resilience Act Conformity (CRA)
- update dependencies

htf-filters-4.0.5

- add support for Python 3.13

htf-filters-4.0.4

- add requirements for validation purposes

htf-filters-4.0.3

- add build environment data for validation purposes

htf-filters-4.0.2

- `htf-filters` can now be used standalone

htf-filters-4.0.1

- add support for Python 3.12

htf-filters-4.0.0

- remove all camel-case methods
- add type-hints
- use `hlm-3.1`

htf-filters-3.0.1

- add support for Python 3.11

htf-filters-3.0.0

- extract `htf-filters` from `htf`

Terminal Colors

HILSTER Testing Framework support terminal colors.

Terminal colors are automatically enabled if the terminal supports it.

They can be disabled by setting the environment variable `HTF_NO_COLORS` to any value.

Colored terminal output can be forced by setting the environment variable `HTF_FORCE_COLORS` to any value.

To print colored text use the methods supplied by `htf.termcolors`.

Example:

```
import htf.termcolors as tc

print(tc.yellow("WARNING: this text an error in yellow"))
print(tc.red("ERROR: this text an error in red"))
print(tc.white_on_cyan_bold("HILSTER"))
```

```
import htf.termcolors as tc

tc.demo()
```

```
htf.termcolors.blue(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'blue'. Background color: None. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'blue'. Background color: None. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'blue'. Background color: None. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_on_cyan(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'blue'. Background color: 'on_cyan'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_on_cyan_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'blue'. Background color: 'on_cyan'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_on_cyan_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'blue'. Background color: 'on_cyan'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_on_cyan_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'blue'. Background color: 'on_cyan'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_on_green(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'blue'. Background color: 'on_green'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_on_green_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'blue'. Background color: 'on_green'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_on_green_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'blue'. Background color: 'on_green'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:

`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_on_green_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'blue'. Background color: 'on_green'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_on_grey(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'blue'. Background color: 'on_grey'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_on_grey_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'blue'. Background color: 'on_grey'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_on_grey_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'blue'. Background color: 'on_grey'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_on_grey_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'blue'. Background color: 'on_grey'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_on_magenta(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'blue'. Background color: 'on_magenta'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_on_magenta_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'blue'. Background color: 'on_magenta'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_on_magenta_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘blue’. Background color: ‘on_magenta’. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_on_magenta_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘blue’. Background color: ‘on_magenta’. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_on_red(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘blue’. Background color: ‘on_red’. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_on_red_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘blue’. Background color: ‘on_red’. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_on_red_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'blue'. Background color: 'on_red'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_on_red_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'blue'. Background color: 'on_red'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_on_white(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'blue'. Background color: 'on_white'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_on_white_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'blue'. Background color: 'on_white'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:

`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_on_white_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'blue'. Background color: 'on_white'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_on_white_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'blue'. Background color: 'on_white'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_on_yellow(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'blue'. Background color: 'on_yellow'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_on_yellow_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'blue'. Background color: 'on_yellow'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_on_yellow_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'blue'. Background color: 'on_yellow'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_on_yellow_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'blue'. Background color: 'on_yellow'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.blue_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'blue'. Background color: None. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: None. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: None. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.critical(text: str) → str
```

Return colored text if supported by terminal using ANSI codes for a critical error.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'cyan'. Background color: None. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'cyan'. Background color: None. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'cyan'. Background color: None. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_on_blue(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'cyan'. Background color: 'on_blue'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_on_blue_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'cyan'. Background color: 'on_blue'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:

`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_on_blue_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'cyan'. Background color: 'on_blue'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_on_blue_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'cyan'. Background color: 'on_blue'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_on_green(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'cyan'. Background color: 'on_green'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_on_green_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'cyan'. Background color: 'on_green'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_on_green_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'cyan'. Background color: 'on_green'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_on_green_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'cyan'. Background color: 'on_green'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_on_grey(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'cyan'. Background color: 'on_grey'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_on_grey_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘cyan’. Background color: ‘on_grey’. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_on_grey_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘cyan’. Background color: ‘on_grey’. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_on_grey_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘cyan’. Background color: ‘on_grey’. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_on_magenta(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘cyan’. Background color: ‘on_magenta’. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_on_magenta_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'cyan'. Background color: 'on_magenta'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_on_magenta_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'cyan'. Background color: 'on_magenta'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_on_magenta_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'cyan'. Background color: 'on_magenta'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_on_red(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'cyan'. Background color: 'on_red'. Attributes: .

Parameters:
text – the text to be colored.
Returns:

`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_on_red_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'cyan'. Background color: 'on_red'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_on_red_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'cyan'. Background color: 'on_red'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_on_red_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'cyan'. Background color: 'on_red'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_on_white(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'cyan'. Background color: 'on_white'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_on_white_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'cyan'. Background color: 'on_white'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_on_white_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'cyan'. Background color: 'on_white'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_on_white_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'cyan'. Background color: 'on_white'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_on_yellow(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'cyan'. Background color: 'on_yellow'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_on_yellow_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'cyan'. Background color: 'on_yellow'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_on_yellow_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'cyan'. Background color: 'on_yellow'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_on_yellow_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'cyan'. Background color: 'on_yellow'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.cyan_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'cyan'. Background color: None. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.debug(text: str) → str
```

Return colored text if supported by terminal using ANSI codes for a critical error.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.demo() → None
```

Print a color demo.

```
htf.termcolors.error(text: str) → str
```

Return colored text if supported by terminal using ANSI codes for an error.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.failure(text: str) → str
```

Return colored text if supported by terminal using ANSI codes for a failure.

Parameters:
text – the text to be colored.
Returns:

`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'green'. Background color: None. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'green'. Background color: None. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'green'. Background color: None. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_on_blue(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'green'. Background color: 'on_blue'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_on_blue_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'green'. Background color: 'on_blue'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_on_blue_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'green'. Background color: 'on_blue'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_on_blue_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'green'. Background color: 'on_blue'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_on_cyan(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘green’. Background color: ‘on_cyan’. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_on_cyan_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘green’. Background color: ‘on_cyan’. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_on_cyan_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘green’. Background color: ‘on_cyan’. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_on_cyan_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘green’. Background color: ‘on_cyan’. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_on_grey(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘green’. Background color: ‘on_grey’. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_on_grey_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘green’. Background color: ‘on_grey’. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_on_grey_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘green’. Background color: ‘on_grey’. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_on_grey_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘green’. Background color: ‘on_grey’. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:

`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_on_magenta(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'green'. Background color: 'on_magenta'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_on_magenta_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'green'. Background color: 'on_magenta'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_on_magenta_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'green'. Background color: 'on_magenta'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_on_magenta_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'green'. Background color: 'on_magenta'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_on_red(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'green'. Background color: 'on_red'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_on_red_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'green'. Background color: 'on_red'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_on_red_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'green'. Background color: 'on_red'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_on_red_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘green’. Background color: ‘on_red’. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_on_white(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘green’. Background color: ‘on_white’. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_on_white_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘green’. Background color: ‘on_white’. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_on_white_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘green’. Background color: ‘on_white’. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_on_white_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'green'. Background color: 'on_white'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_on_yellow(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'green'. Background color: 'on_yellow'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_on_yellow_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'green'. Background color: 'on_yellow'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_on_yellow_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'green'. Background color: 'on_yellow'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:

`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_on_yellow_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'green'. Background color: 'on_yellow'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.green_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'green'. Background color: None. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'grey'. Background color: None. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'grey'. Background color: None. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'grey'. Background color: None. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_on_blue(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'grey'. Background color: 'on_blue'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_on_blue_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'grey'. Background color: 'on_blue'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_on_blue_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘grey’. Background color: ‘on_blue’. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_on_blue_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘grey’. Background color: ‘on_blue’. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_on_cyan(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘grey’. Background color: ‘on_cyan’. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_on_cyan_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘grey’. Background color: ‘on_cyan’. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_on_cyan_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'grey'. Background color: 'on_cyan'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_on_cyan_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'grey'. Background color: 'on_cyan'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_on_green(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'grey'. Background color: 'on_green'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_on_green_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'grey'. Background color: 'on_green'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:

`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_on_green_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'grey'. Background color: 'on_green'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_on_green_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'grey'. Background color: 'on_green'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_on_magenta(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'grey'. Background color: 'on_magenta'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_on_magenta_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'grey'. Background color: 'on_magenta'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_on_magenta_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'grey'. Background color: 'on_magenta'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_on_magenta_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'grey'. Background color: 'on_magenta'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_on_red(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'grey'. Background color: 'on_red'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_on_red_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘grey’. Background color: ‘on_red’. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_on_red_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘grey’. Background color: ‘on_red’. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_on_red_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘grey’. Background color: ‘on_red’. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_on_white(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘grey’. Background color: ‘on_white’. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_on_white_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'grey'. Background color: 'on_white'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_on_white_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'grey'. Background color: 'on_white'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_on_white_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'grey'. Background color: 'on_white'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_on_yellow(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'grey'. Background color: 'on_yellow'. Attributes: .

Parameters:
text – the text to be colored.
Returns:

`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_on_yellow_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'grey'. Background color: 'on_yellow'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_on_yellow_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'grey'. Background color: 'on_yellow'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_on_yellow_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'grey'. Background color: 'on_yellow'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.grey_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'grey'. Background color: None. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.info(text: str) → str
```

Return colored text if supported by terminal using ANSI codes for an information.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘magenta’. Background color: None. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘magenta’. Background color: None. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘magenta’. Background color: None. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_on_blue(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘magenta’. Background color: ‘on_blue’. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_on_blue_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘magenta’. Background color: ‘on_blue’. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_on_blue_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘magenta’. Background color: ‘on_blue’. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_on_blue_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'magenta'. Background color: 'on_blue'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_on_cyan(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'magenta'. Background color: 'on_cyan'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_on_cyan_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'magenta'. Background color: 'on_cyan'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_on_cyan_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'magenta'. Background color: 'on_cyan'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:

`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_on_cyan_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'magenta'. Background color: 'on_cyan'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_on_green(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'magenta'. Background color: 'on_green'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_on_green_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'magenta'. Background color: 'on_green'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_on_green_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'magenta'. Background color: 'on_green'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_on_green_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'magenta'. Background color: 'on_green'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_on_grey(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'magenta'. Background color: 'on_grey'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_on_grey_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'magenta'. Background color: 'on_grey'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_on_grey_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘magenta’. Background color: ‘on_grey’. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_on_grey_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘magenta’. Background color: ‘on_grey’. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_on_red(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘magenta’. Background color: ‘on_red’. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_on_red_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘magenta’. Background color: ‘on_red’. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_on_red_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'magenta'. Background color: 'on_red'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_on_red_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'magenta'. Background color: 'on_red'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_on_white(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'magenta'. Background color: 'on_white'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_on_white_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'magenta'. Background color: 'on_white'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:

`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_on_white_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'magenta'. Background color: 'on_white'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_on_white_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'magenta'. Background color: 'on_white'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_on_yellow(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'magenta'. Background color: 'on_yellow'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_on_yellow_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'magenta'. Background color: 'on_yellow'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_on_yellow_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'magenta'. Background color: 'on_yellow'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_on_yellow_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'magenta'. Background color: 'on_yellow'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.magenta_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'magenta'. Background color: None. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.make_style(color: str | None = None, on_color: str | None = None, attrs:
List[str] | None = None) → Callable[[str], str]
```

Return a callable that has a parameter text that will be colored according to the parameters if coloring is supported.

Parameters:

- **color=None** – the color for the text
- **on_color=None** – the background color
- **attrs=None** – a list of attributes

```
htf.termcolors.on_blue(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: 'on_blue'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_blue_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: 'on_blue'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_blue_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: 'on_blue'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.

Return type:
str

```
htf.termcolors.on_blue_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: 'on_blue'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_cyan(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: 'on_cyan'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_cyan_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: 'on_cyan'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_cyan_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: 'on_cyan'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_cyan_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: 'on_cyan'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_green(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: 'on_green'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_green_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: 'on_green'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_green_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: 'on_green'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_green_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: 'on_green'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_grey(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: 'on_grey'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_grey_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: 'on_grey'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_grey_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: ‘on_grey’. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_grey_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: ‘on_grey’. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_magenta(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: ‘on_magenta’. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_magenta_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: ‘on_magenta’. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:

`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_magenta_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: 'on_magenta'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_magenta_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: 'on_magenta'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_red(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: 'on_red'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_red_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: 'on_red'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_red_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: 'on_red'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_red_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: 'on_red'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_white(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: 'on_white'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_white_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: ‘on_white’. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_white_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: ‘on_white’. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_white_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: ‘on_white’. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_yellow(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: ‘on_yellow’. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_yellow_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: ‘on_yellow’. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_yellow_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: ‘on_yellow’. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.on_yellow_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: ‘on_yellow’. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘red’. Background color: None. Attributes: .

Parameters:
text – the text to be colored.
Returns:

`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'red'. Background color: None. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'red'. Background color: None. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_on_blue(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'red'. Background color: 'on_blue'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_on_blue_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'red'. Background color: 'on_blue'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_on_blue_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'red'. Background color: 'on_blue'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_on_blue_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'red'. Background color: 'on_blue'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_on_cyan(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'red'. Background color: 'on_cyan'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_on_cyan_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘red’. Background color: ‘on_cyan’. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_on_cyan_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘red’. Background color: ‘on_cyan’. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_on_cyan_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘red’. Background color: ‘on_cyan’. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_on_green(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘red’. Background color: ‘on_green’. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_on_green_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'red'. Background color: 'on_green'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_on_green_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'red'. Background color: 'on_green'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_on_green_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'red'. Background color: 'on_green'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_on_grey(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'red'. Background color: 'on_grey'. Attributes: .

Parameters:
text – the text to be colored.
Returns:

`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_on_grey_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'red'. Background color: 'on_grey'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_on_grey_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'red'. Background color: 'on_grey'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_on_grey_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'red'. Background color: 'on_grey'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_on_magenta(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'red'. Background color: 'on_magenta'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_on_magenta_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'red'. Background color: 'on_magenta'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_on_magenta_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'red'. Background color: 'on_magenta'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_on_magenta_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'red'. Background color: 'on_magenta'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_on_white(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘red’. Background color: ‘on_white’. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_on_white_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘red’. Background color: ‘on_white’. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_on_white_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘red’. Background color: ‘on_white’. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_on_white_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘red’. Background color: ‘on_white’. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_on_yellow(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘red’. Background color: ‘on_yellow’. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_on_yellow_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘red’. Background color: ‘on_yellow’. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_on_yellow_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘red’. Background color: ‘on_yellow’. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_on_yellow_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘red’. Background color: ‘on_yellow’. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:

`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.red_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'red'. Background color: None. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.skipped(text: str) → str
```

Return colored text if supported by terminal using ANSI codes for something skipped.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.success(text: str) → str
```

Return colored text if supported by terminal using ANSI codes for a success.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.system(text: str) → str
```

Return colored text if supported by terminal using ANSI codes for system message.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:

str

```
htf.termcolors.underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: None. Background color: None. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.warning(text: str) → str
```

Return colored text if supported by terminal using ANSI codes for a failure.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘white’. Background color: None. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘white’. Background color: None. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:

`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'white'. Background color: None. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_on_blue(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'white'. Background color: 'on_blue'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_on_blue_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'white'. Background color: 'on_blue'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_on_blue_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'white'. Background color: 'on_blue'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_on_blue_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'white'. Background color: 'on_blue'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_on_cyan(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'white'. Background color: 'on_cyan'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_on_cyan_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'white'. Background color: 'on_cyan'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_on_cyan_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘white’. Background color: ‘on_cyan’. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_on_cyan_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘white’. Background color: ‘on_cyan’. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_on_green(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘white’. Background color: ‘on_green’. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_on_green_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘white’. Background color: ‘on_green’. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_on_green_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'white'. Background color: 'on_green'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_on_green_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'white'. Background color: 'on_green'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_on_grey(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'white'. Background color: 'on_grey'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_on_grey_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'white'. Background color: 'on_grey'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:

`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_on_grey_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'white'. Background color: 'on_grey'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_on_grey_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'white'. Background color: 'on_grey'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_on_magenta(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'white'. Background color: 'on_magenta'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_on_magenta_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'white'. Background color: 'on_magenta'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_on_magenta_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'white'. Background color: 'on_magenta'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_on_magenta_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'white'. Background color: 'on_magenta'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_on_red(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'white'. Background color: 'on_red'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_on_red_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘white’. Background color: ‘on_red’. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_on_red_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘white’. Background color: ‘on_red’. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_on_red_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘white’. Background color: ‘on_red’. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_on_yellow(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘white’. Background color: ‘on_yellow’. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_on_yellow_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'white'. Background color: 'on_yellow'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_on_yellow_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'white'. Background color: 'on_yellow'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_on_yellow_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'white'. Background color: 'on_yellow'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.white_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'white'. Background color: None. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:

`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'yellow'. Background color: None. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'yellow'. Background color: None. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'yellow'. Background color: None. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_on_blue(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'yellow'. Background color: 'on_blue'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_on_blue_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'yellow'. Background color: 'on_blue'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_on_blue_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'yellow'. Background color: 'on_blue'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_on_blue_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'yellow'. Background color: 'on_blue'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_on_cyan(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘yellow’. Background color: ‘on_cyan’. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_on_cyan_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘yellow’. Background color: ‘on_cyan’. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_on_cyan_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘yellow’. Background color: ‘on_cyan’. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_on_cyan_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: ‘yellow’. Background color: ‘on_cyan’. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_on_green(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'yellow'. Background color: 'on_green'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_on_green_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'yellow'. Background color: 'on_green'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_on_green_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'yellow'. Background color: 'on_green'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_on_green_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'yellow'. Background color: 'on_green'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:

`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_on_grey(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'yellow'. Background color: 'on_grey'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_on_grey_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'yellow'. Background color: 'on_grey'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_on_grey_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'yellow'. Background color: 'on_grey'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_on_grey_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'yellow'. Background color: 'on_grey'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_on_magenta(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'yellow'. Background color: 'on_magenta'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_on_magenta_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'yellow'. Background color: 'on_magenta'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_on_magenta_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'yellow'. Background color: 'on_magenta'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_on_magenta_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'yellow'. Background color: 'on_magenta'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_on_red(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'yellow'. Background color: 'on_red'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_on_red_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'yellow'. Background color: 'on_red'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_on_red_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'yellow'. Background color: 'on_red'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_on_red_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'yellow'. Background color: 'on_red'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_on_white(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'yellow'. Background color: 'on_white'. Attributes: .

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_on_white_bold(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'yellow'. Background color: 'on_white'. Attributes: bold.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_on_white_bold_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'yellow'. Background color: 'on_white'. Attributes: bold, underline.

Parameters:
text – the text to be colored.
Returns:

`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_on_white_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'yellow'. Background color: 'on_white'. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

```
htf.termcolors.yellow_underline(text)
```

Return colored text if supported by terminal using ANSI codes.

Foreground color: 'yellow'. Background color: None. Attributes: underline.

Parameters:
text – the text to be colored.
Returns:
`text` colored if supported by the terminal.
Return type:
str

Mixins

Mixins can be used to import code without inheritance.

Delegation

The *Delegation* mixins give the possibility to automatically delegate to members of a class without calling set() or get() directly.

Delegation is used for hardware abstractors.

```
class htf.mixins.DelegationError
```

`DelegationError` is thrown if `DelegateMixin` is used directly without being mixed in.

```
class htf.mixins.DelegateMixin
```

`DelegateMixin` defines `set` and `get` as methods that have to be overwritten by subclass.

```
get() → Any
```

The getter for a delegate.

Note

This method is abstract.

```
set(value: Any) → None
```

The setter for a delegate.

Parameters:
value – the value to be set.

Note

This method is abstract.

```
class htf.mixins.DelegateCallbackMixin(set_callback: Callable, get_callback: Callable)
```

`DelegateCallbackMixin` gives the possibility to user define setter and getter methods using a `DelegateMixin`.

Parameters:

- **set_callback** – the method to be called as a setter.
- **get_callback** – the method to be called as a getter.

```
class htf.mixins.DelegatorMixin
```

The `DelegatorMixin` automatically delegates calls for `DelegateMixin.set` and `DelegateMixin.get` to its members if they are instances of `DelegateMixin`.

Example usage:

```
class ConcreteDelegate(DelegateMixin):

    def __init__(self, parameter):
        self.parameter = parameter
        self.value = None
```

```
    def set(self, value):
        print(self.__class__.__name__ + ".set()", value)
        self.value = value

    def get(self):
        print(self.__class__.__name__ + ".get()")
        return self.value

class AnotherDelegate(ConcreteDelegate):
    pass

class A(DelegatorMixin):

    def __init__(self):
        self.member = ConcreteDelegate("parameter for member")
        self.test = AnotherDelegate("blub")

class NormalMember:

    def __init__(self, parameter):
        self.parameter = parameter
        self.value = None

    def set(self, value):
        print(self.__class__.__name__ + ".set()", value)
        self.value = value

    def get(self):
        print(self.__class__.__name__ + ".get()")
        return self.value

a = A()
print(a.member)
a.member = 1
print(a.member)
a.b = 2
print(a.b)
a.test = 1337
print(a.test)
a.blub = AnotherDelegate("rolf") # a new delegate is instantiated
a.blub = "peter" # now setting a value leads to a delegated call to AnotherDelegate().set("peter")
print(a.blub) # new reading is delegated to it
a.normalMember = NormalMember("normal")
a.normalMember = "test"
print(a.normalMember)
```

Checksums

Checksum consists of `htf.checksum.CRC`.

CRC

`htf.checksum.CRC` allows users to easily calculate crc checksums over data.

```
class htf.checksum.CRC(width: int, polynomial: int, reflect_input: bool, initial_value: int,
reflect_output: bool, xor_output: int)
```

A configurable CRC calculator.

The CRC constructor.

Parameters:

- **width** – the CRC's width in bits
- **polynomial** – the CRC's polynomial
- **reflect_input** – if set to `True` input is reflected
- **initial_value** – the initial value
- **reflect_output** – if set to `True` output is reflected
- **xor_output** – xor all outputs with `xor_output`

```
calculate(data: str | bytes | List[int]) → int
```

Calculates the crc over `data` bit by bit.

Parameters:
data – the input data to be used for the crc calculation
Returns:
the calculated crc for `data`
Return type:
int

Example

This example calculates a CRC-16 over `"123456789"`.

```
from htf.checksum import CRC

crc16 = CRC(width=16,
            polynomial=0x8005,
            reflect_input=True,
            initial_value=0,
            reflect_output=True,
            xor_output=0)

result = crc16.calculate(b"123456789")
assert result == 0xBB3D
```

Changelog

htf-checksum-3.2.0

- ship `cp312-abi3` wheels that are compatible with Python 3.12 and newer versions

htf-checksum-3.1.6

- implement Cyber Resilience Act Conformity (CRA)
- update dependencies

htf-checksum-3.1.5

- add support for Python 3.13

htf-checksum-3.1.4

- add requirements for validation purposes

htf-checksum-3.1.3

- add build environment data for validation purposes

htf-checksum-3.1.2

- `htf-checksum` can now be used standalone

htf-checksum-3.1.1

- add support for Python 3.12

htf-checksum-3.1.0

- add type-hints
- use `hlm-3.1`

htf-checksum-3.0.1

- add support for Python 3.11

htf-checksum-3.0.0

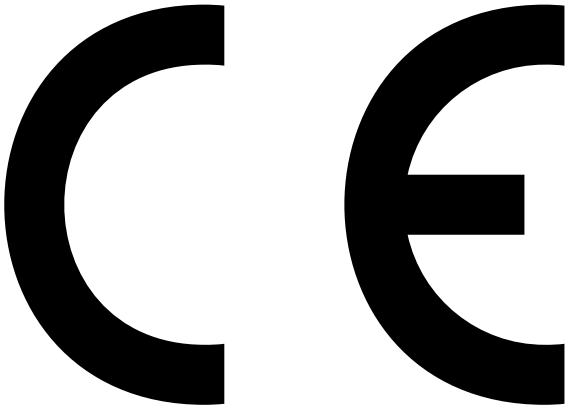
- extract `htf-checksum` from `htf`

Cyber Resilience Act Conformity

This software is distributed by

HILSTER Testing Solutions GmbH
Amtsstraße 10
22143 Hamburg
Germany

Statement of Conformity



This software is placed on the market as a product with digital elements in the sense of Regulation (EU) 2024/2847 (Cyber Resilience Act, “CRA”). It is therefore subject to the conformity assessment obligations of the CRA.

This software is classified as “standard” risk class under the CRA.

Security Measures

As required by the CRA, HILSTER Testing Solutions GmbH takes the following measures to ensure a high level of security:

- Regular review of the source code and the dependencies used
- Prompt provision of updates for known security vulnerabilities
- Minimised collection and storage of user data
- No telemetry or automatic transmission of data to third parties

Reporting Vulnerabilities

Security vulnerabilities can be reported at any time by sending an e-mail to security@hilster.io.

> **Note**
> This statement is for information purposes only and does not constitute a declaration of conformity within the meaning of the Cyber Resilience Act.

Changelog

htf-4.3.1

- update dependencies

htf-4.3.0

- ship `cp312-abi3` wheels that are compatible with Python 3.12 and newer versions
- update installation instructions
- ISO/TR 80002-2 certification is being discontinued

> **Note**
>
> Compliance Update: Formal ISO/TR 80002-2 certification is suspended until further notice to prioritize the full implementation of the Cyber Resilience Act (CRA) regulations. Eliminating regulatory redundancies allows us to focus development resources on upcoming legal cybersecurity requirements.

htf-4.2.6

- implement Cyber Resilience Act Conformity (CRA)
- update dependencies

htf-4.2.5

- update dependencies
- use latest hlm
- fix a bug that showed `None`'s docstring as the tests's docstring
- clarify wording in End-User License Agreement (EULA) for htf-community

htf-4.2.4

- add support for Python 3.14

htf-4.2.3

- automatically enable colored terminal output in GitLab CI/CD
- add an option to force colored terminal output

htf-4.2.2

- add support for Xentara

htf-4.2.1

- add support for Python 3.13
- improve automatic update check

htf-4.2.0

- add support for Matrix Requirements ALM/QMS
- Behavior Driven Development now supports repeated keywords to implement complex tests with BDD
- add `htf.implements` and the `Implements:` keyword for Behavior Driven Development
- Replace some internal dependencies with Rust-based ones
- `htf` checks for available updates. See documentation for more information.
- add `htf.get_fixture` and `htf.get_fixture_sync` to access or instantiate fixtures outside tests
- `htf.fixtures.step` can be used as a `dict` now to report actual results of a step, for example
- fix an exception-related bug in generator fixtures leading to test reports with draft status
- improve rendering of metadata in HTML test report

htf-4.1.3

- htf-community now supports recent and supported Python versions with status `feature` or `bugfix`

htf-4.1.2

- fix a bug with steps and Behavior Driven Development tests in interactive tests

htf-4.1.1

- fix unlocking of explicit features

htf-4.1.0

- htf is certified for ISO/TR 80002-2: Medical device software - Part 2: Validation of software for medical device quality systems. See Certifications documentation for more information.

Note

ISO/TR 80002-2 applies to any software used in device design, testing, component acceptance, manufacturing, labelling, packaging, distribution and complaint handling or to automate any other aspect of a medical device quality system as described in ISO 13485.

ISO/TR 80002-2 applies to

- software used in the quality management system,
- software used in production and service provision, and
- software used for the monitoring and measurement of requirements.

It does not apply to

- software used as a component, part or accessory of a medical device, or
- software that is itself a medical device.

htf-4.0.4

- improve handling of module names for discovered files
- allow relative imports in test files
- fix a rare edge case during test discovery where tests were collected twice

htf-4.0.3

- use `docutils==0.20.1`

htf-4.0.2

- add support for Python 3.12

htf-4.0.1

- add support for LabJack DAQ devices in `htf-io==4.1.0`

htf-4.0.0

- add complete support for asyncio for tests, fixtures, steps, etc.
- add type hints
- remove all legacy code
- improve assertion formatting
- improve exception formatting
- add options to not open the browser automatically for Interactive Testing.
- add async `htf.fixtures.sleep` fixture

- `htf.retry` is a coroutine now and `htf.retry_sync` can be used instead to be backward compatible
- update all htf-module versions
- update dependencies

> **Warning**
>
> htf-4 is not fully backwards compatible to htf-3!
>
> - drop support for Python 3.7 and Python 3.8 (Raspberry Pi OS uses Python 3.9 now so this is not needed anymore)
> - PEP 8: all names are PEP-8-ified and camel-case aliases have been removed
> - some decorators now only return coroutines (but if they are only used as documented you will not have to change anything)
> - remote `logo` parameter from HTML test report
> - the parameter names for `htf.periodic` changed
> - `htf.TestCase` now expects set up and tear down methods in snake case: `set_up()`, `set_up_class`, `tear_down` and `tear_down_class` to be compatible to PEP 8
> - all camel case assertion methods were removed from `htf.TestCase` to be compatible to PEP 8
> - `failfast: bool` and `--failfast` have been renamed to `fail_fast: bool` and `--fail-fast`

htf-3.0.6

> **Warning**
>
> htf-3.0 reached the end of life. Please upgrade to htf-4.0 which can be used with little changes.

- update dependencies to fix installation problems

htf-3.0.5

- add support for Python 3.11
- add @metadata decorator

htf-3.0.4

- set test result before tearing down fixtures

htf-3.0.3

- fix exception handling with incorrectly formatted docstrings in HTML test report

- fix types in docstrings

htf-3.0.2

- improve exception handling in generator fixtures

htf-3.0.1

- add skip reasons to HTML Test Report Overview chart
- parameters is initialized with an empty `dict` instead of `None` if no parameters are supplied
- data driven testing generators are reset and thus can be run multiple times
- add start time, duration, etc. to draft reports
- open reports with absolute filenames to support Microsoft's browsers Edge and Internet Explorer when using `htf.main(..., open_reports=True)` or `htf --open-report`
- test result is `error` if `htf.SkipStep` is raised outside a test step

htf-3.0.0

> **Warning**
>
> htf-3 drops support for Python 2.7 and 32 bit software!
>
> This is why:
>
> - We offered Python 2.7 support for 2 years after Python 2.7's end of life on 2020-01-01.
> - No recent computer has a 32 bit processor anymore.
> - We will support the newest and commonly used versions of Python 3.x.

- `htf-3` adds support for Behavior Driven Development including tagging and requirements coverage
- add support for parameters to supply run-time parameters from command-line
- add support for overwriting settings from command-line to change settings at run-time
- update look and feel of QABench
- update and modernize HTML Test Report
- installation has changed and `pip install -i https://pypi.hilster.io htf` only installs the core of `htf` use `pip install -i https://pypi.hilster.io htf[full]` for the behavior of older versions and get more information in the installation instructions and installation options
- `htf` is split into multiple wheels (some imports changed)
- `htf -s` to minimize report size is now `htf -M`

- `htf-3` uses `oser-3` (needs a few changes - see oser's changelog)
- add a configurable timeout to power supplies
- add support for `pip-21`

htf-2.0.18

> **Warning**
> htf-2.0 reached the end of life on January 1st, 2022. Please upgrade to htf-3.0 which can be used with little changes.

- update look and feel of QABench
- update and modernize HTML Test Report
- make tags generated by data driven tests usable
- fix `htf.skip_if` when using callable conditions defined in classes

htf-2.0.17

- use `hlm-2.0.1`

htf-2.0.16

- Modbus TCP transaction identifier can overflow now
- add an exception if Modbus TCP receives an unexpected response
- convert all test run metadata's values to strings

htf-2.0.15

- fixtures are not instantiated if a test is skipped using `htf.skip` or `htf.skip_if`
- fix generation of test steps in `htf.DOORSTestReport`

htf-2.0.14

- force installation of `docutils==0.16`

htf-2.0.13

- fix tear down order of dependant fixtures
- add support for ethernet, general output as well as storing and recalling instrument states for the Rohde + Schwarz HMP Series programmable power supplies

htf-2.0.12

- fix HTML report loading for empty mimetypes in attachments and empty metadata

htf-2.0.11

- use `docutils==0.16`

htf-2.0.10

- fix evaluation crash when exceptions contained private use unicode characters
- fix error sometimes thrown in `htf.fixtures.delay` fixture

htf-2.0.9

- use `hlm-2.0`

htf-2.0.8

- fix usage of tags in combination with fixture tags
- fix fixture cleanup
- improve interaction user interface and fix a bug that caused it to freeze

htf-2.0.7

- reduce HTML report size
- improve stability and user experience of interaction user interface
- improve stability of test discovery
- fix disappearing steps in HTML report
- fix tags inheritance when using classes
- fix Modbus client's query method thread-safeness
- fix message in `htf.assertions.assert_list_equal`
- fix rarely occurring escaping error in JUnit XML report
- update to `pyserial==3.5`

htf-2.0.6

- use Dashboard report server url as is to support `Dashboard-1.0`
- add -o option to `report-tool` to open generated HTML reports
- fix date in HTML Test Report
- improve format of JUnit-XML-Testreport
- fix report merge errors with `report-tool`
- add support for Python 3.9
- update dependencies

htf-2.0.5

- fix docstring rendering in HTML Test Report's print view

htf-2.0.4

- fix threads fixture's `htf.fixtures.threads.stop` method
- `htf.fixtures.step` supports `*args` and `**kwargs`
- add `htf.fixtures.step.skip` to skip steps
- fix `htf.fixtures.jira` fixture
- fix printed list of collected fixtures
- fix some bugs on Python 2.7

htf-2.0.3

- fix shuffle

htf-2.0.2

- add @requirements decorator
- fix total test run time in output to correct display days
- update interfacing with the Dashboard

htf-2.0.1

- fix crash in Report Tool when converting from XML source

htf-2.0.0

- HTML Test Report is interactive now and is printable
- Completely renewed the internal test runner
- Interactive Testing
- Fixtures
- Report Tool
- Draft Reports in htf and htf.main()
- Stoppable Threads
- Tests can be shuffled in htf and htf.main()
- Test Steps can be hierarchic
- Reduce wheel size

> **Warning**
>
> htf-2 is not fully backwards compatible to htf-1!
>
> - PEP 8: all names are PEP-8-ified but have aliases for backwards compatibility. Some parameter names changed and you will have to change them yourself.
> - Verbosity: The `verbosity` parameter and `--verbosity` have been removed.
> - Catchbreak: The `catchbreak` parameter and `--catchbreak` have been removed. `catchbreak` is enabled by default.
> - TestCase: `_runThreadsEnable` was removed. Use the threads-fixture.

htf-1.3.14

> **Warning**
>
> htf-1.3 reached the end of life on August 15th, 2021. Please upgrade to htf-2.0 which can be used with little changes.

- use `docutils==0.16`

htf-1.3.13

- remove docs from wheels

htf-1.3.12

- use `hlm-2.0`
- fix Modbus client's query method thread-safeness

htf-1.3.11

- add support for Python 3.9
- fix tags inheritance when using classes
- fix Modbus client's query method thread-safeness
- fix rarely occurring escaping error in JUnit XML report
- update to `pyserial==3.5`

htf-1.3.10

- use Dashboard report server url as is to support `Dashboard-1.0`

htf-1.3.9

- fix HTTP status code for new Dashboard API

htf-1.3.8

- fix `htf` console entry point to enable networking in Linux

htf-1.3.7

- fix payload length calculation for HART command 3 and 61
- fix object clean up order in HartInterface and HartApplicationLayer
- fix potential element ordering problem in CSVFileIterator

htf-1.3.6

- build manylinux2014 wheels for Python 3.x and manylinux2010 wheels for Python 2.7

htf-1.3.5

- Serial Modbus clients and servers now accept additional serial parameters

htf-1.3.4

- `OserPrompt` now allows NaN for float values

htf-1.3.3

- fix `hlm` usage

htf-1.3.2

- > **Warning**
  >
  > Due to some bugfixes in hlm the host id might change. We're sorry. Please renew your license.

- add support for Python 3.8 (htf is shipped for Python 2.7, 3.6, 3.7 and 3.8 for Windows and Linux for 32 and 64 bit machines)
- update to `hlm-1.1`
- update to `oser~=1.3.3`
- fix HART `Time` encoding and decoding
- add HART response code `error_command_not_implemented`
- improve HART on Linux
- fix Modbus client debug
- add interactive mode for htf in Docker (see Interactive Mode)
- fix report server report and report upload

- improve docs for `skip_test` and `skip_step`

htf-1.3.1

- update to hlm-1.0.4
- update to oser-1.3.1

htf-1.3.0

- htf runs in Docker (see Docker Integration) on Linux and Windows
- add support for Data Driven Testing
- add a full-stack Modbus — The Modbus Testing Framework implementation
- add support for the Report Server (see htf.main() — Testscript utility and htf — Command line utility)
- completely revamped Settings and add support for settings with YAML and for default values
- add a YAML test report (see htf.main() — Testscript utility and htf — Command line utility)
- add an option to keep captured stdout and stderr only in case of a failed test (see htf.main() — Testscript utility and htf — Command line utility)
- improve console output for htf.main() — Testscript utility and htf — Command line utility
- nonexistent folders for test reports are created automatically