TECHNICAL DOCUMENTATION

OSER

Complete documentation.
One clear reference.

Version 3.3.0

Contents

OSER — Objectoriented Declarative Parser and Builder for Binary Data **5**

Installation **5**

Installation with pip 5

Check Installation 6

Upgrading 6

Installing a Specific Version 6

Support **6**

Overview **7**

Introduction 7

General Concepts 7

OserNode 9

Accessing Data 11

The Context 13

Copying 17

Examples **17**

EEPROM-Content 17

Custom Network Protocol 19

HTTP-Request 24

Access Members By Name 32

Primitive Types **32**

ByteType 32

Naming Conventions 35

8-Bit Types 35

16-Bit Types 47

32-Bit Types 57

64-Bit Types 67

Variable Length Integers 77

Float Types 89

Double Types 94

BitType 99

Bit Types 101

Null 111

Nothing 114

Complex Types **116**

ByteStruct 116

BitStruct 119

Enum 122

Context Based Types **127**

The Context 127

String 127
Data 132
Array 134
If 141
IfElse 145
Switch 149
Value 154

Checksum Types 156
CRC — Cyclic Redundancy Check 156

Special Types 189
Constant 189
Google Protocol Buffers Adapter 190
JSON Adapter 196
Regular Expression Match 199
Repeat Until 200
Lazy Initialization 204
BitMap 205
Modbus Types 209

Rich Comparison 233

Acceleration 234

Fuzzing 234
Concepts 234
Deep Copies 238
Fuzzing Random Data Adapters 239
Special Elements 244
Mixins 249
Examples 250
Full Fuzzing Example 254

Utilities 256

Cyber Resilience Act Conformity 256
Statement of Conformity 256
Security Measures 257
Reporting Vulnerabilities 257

Changelog 257
oser-3.3.0: 257
oser-3.2.0: 257
oser-3.1.6: 257
oser-3.1.5: 257
oser-3.1.4: 257
oser-3.1.3: 258
oser-3.1.2: 258
oser-3.1.1: 258
oser-3.1.0: 258
oser-3.0.1: 258
oser-3.0.0: 258

oser-2.x ... 258
oser-1.3.10 ... 259
oser-1.3.9 ... 259
oser-1.3.8 ... 259
oser-1.3.7 ... 259
oser-1.3.6 ... 259
oser-1.3.5 ... 259
oser-1.3.4: ... 259
oser-1.3.3: ... 259
oser-1.3.2: ... 260
oser-1.3.1: ... 260
oser-1.3.0: ... 260
oser-1.2.9: ... 260
oser-1.2.8: ... 261
oser-1.2.7: ... 261
oser-1.2.6: ... 261
oser-1.2.5: ... 261
oser-1.2.4: ... 261
oser-1.2.3: ... 261
oser-1.2.2: ... 261
oser-1.2.1: ... 261
oser-1.2.0: ... 261
oser-1.1.13: ... 262
oser-1.1.12: ... 262
oser-1.1.11: ... 262
oser-1.1.10: ... 262
oser-1.1.9: ... 262
oser-1.1.8: ... 262
oser-1.1.7: ... 263
oser-1.1.6: ... 263
oser-1.1.5: ... 263
oser-1.1.4: ... 263
oser-1.1.3: ... 263
oser-1.1.2: ... 263
oser-1.1.1: ... 263
oser-1.1.0: ... 263
oser-1.0.6: ... 264
oser-1.0.5: ... 264
oser-1.0.4: ... 265
oser-1.0.3: ... 265
oser-1.0.2: ... 265
oser-1.0.1: ... 265
oser-1.0.0: ... 265
Indices and tables ... 266

OSER — Objectoriented Declarative Parser and Builder for Binary Data

Installation

You should use a recent version of Python 3. Python 2.7 is still supported.

If you need to download Python visit the Python download page.

On Windows you should install it into the recommended path (on Windows this is typically `C:\Python39\`) and add `C:\Python39\Scripts\` to the `%PATH%` environment variable to be able to run `pip` from commandline.

Installation with pip

Simply run:

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

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

You need to set `https_proxy` and `http_proxy` to `http(s)://<username>:<password>@<host>:<port>`.

On Linux enter:

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

```
export http_proxy=http://<username>:<password>@<host>:<port>
```

and on Windows enter:

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

```
set http_proxy=http://<username>:<password>@<host>:<port>
```

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

Alternatively you can use the `--proxy` option:

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

Check Installation

```
python -c "import hlm"
python -c "import htf"
python -c "import oser"
```

If the program quits quietly you're done. If you need to obtain a license read further.

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 -U htf==1.0.0
```

for example.

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

Support

Don't panic — we can help you!

To contact our support team simply run

```
support
```

from the command line and follow the instructions.

Overview

Introduction

OSER is an easy to use, flexible object oriented serializer and deserializer that can be used for translation between binary data formats and its internal structure represented by python classes.

OSER is like a toolbox. It offers many building blocks to build your data format quickly and easily.

Unlike other serializers OSER makes it possible to inspect your data instances in an hierarchical way with or without the binary data aligned to the members. In addition OSER ist capable of building conditional serializers and deserializers using `oser.IfElse`, `oser.Switch`, `oser.Array` and `oser.String` by accessing the context.

For example OSER can be used to build and parse network protocols, image data and various binary files or you can create the content for an EEPROM for an embedded system, etc..

General Concepts

An OSER instance consists of a tree of other OSER instances that implement `oser.OserNode`. Every OSER instance sublasses the `oser.OserNode`. The composite pattern is used here.

Every member that does not start with `_` is included in serialization and deserialization.

```
>>> from oser import ByteStruct, UBInt8, UBInt16, to_hex

>>> class SubData(ByteStruct):
...     def __init__(self):
...         super(SubData, self).__init__()
...         self.sub1 = UBInt8(1)
...         self.sub2 = UBInt16(1000)
...

>>> class Data(ByteStruct):
...     def __init__(self):
...         super(Data, self).__init__()
...         self.a = UBInt8(1)
...         self.b = UBInt16(1000)
...         self.sub = SubData()
...
>>> instance = Data()
```

Every OSER instance can be viewed when it is converted into a string, for example:

```
>>> print(instance)
Data():
    a: 1 (UBInt8)
```

```
    b: 1000 (UBInt16)
    sub: SubData():
        sub1: 1 (UBInt8)
        sub2: 1000 (UBInt16)
```

This shows the internal structure only. To view the internal structure and the binary structure, you can use `oser.OserNode.introspect()`:

```
>>> print(instance.introspect())
   -    -  Data():
   0 \x01      a: 1 (UBInt8)
   1 \x03      b: 1000 (UBInt16)
   2 \xe8
   -    -      sub: SubData():
   3 \x01          sub1: 1 (UBInt8)
   4 \x03          sub2: 1000 (UBInt16)
   5 \xe8
```

To build binary data from an OSER instance, simply use `oser.OserNode.encode()`:

```
>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3|  4|  5
\x01\x03\xE8\x01\x03\xE8
```

To decode binary data into an OSER instance, simply use `oser.OserNode.decode()`:

```
>>> data = b"\x01\x02\x03\x04\x05\x06"
>>> bytes_decoded = instance.decode(data)
>>> print(bytes_decoded)
6
>>> print(instance.introspect())
   -    -  Data():
   0 \x01      a: 1 (UBInt8)
   1 \x02      b: 515 (UBInt16)
   2 \x03
   -    -      sub: SubData():
   3 \x04          sub1: 4 (UBInt8)
   4 \x05          sub2: 1286 (UBInt16)
   5 \x06
```

To access the root element call `oser.OserNode.root()`:

```
>>> root = instance.sub.sub2.root()
>>> print(root)
Data():
    a: 1 (UBInt8)
    b: 515 (UBInt16)
    sub: SubData():
```

```
        sub1: 4 (UBInt8)
        sub2: 1286 (UBInt16)
```

To access the upper element (parent element) call `oser.OserNode.up()`:

```
>>> up = instance.sub.sub2.up()
>>> print(up)
SubData():
    sub1: 4 (UBInt8)
    sub2: 1286 (UBInt16)


>>> upup = up.up()
>>> print(upup)
Data():
    a: 1 (UBInt8)
    b: 515 (UBInt16)
    sub: SubData():
        sub1: 4 (UBInt8)
        sub2: 1286 (UBInt16)
```

Navigation in the element tree is useful to build conditional blocks.

OserNode

oser.core.OserNode

```
class oser.OserNode
```

```
__str__(indent: int = 0, name: str | None = None, stop_at: ByteStruct | BitStruct |
ByteType | BitType | None = None) → str
```

Return the representation of the object as a string.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string into a byte type 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_size() → int
```

Return the size in bytes or bits depending on the type.

```
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 the root element.

```
up() → ByteStruct | BitStruct
```

Return the parent element.

Accessing Data

Accessing Members

To access members in OSER instances simply use the `dot`. Your IDE is able to help you expanding the valid values. In eclipse you can press `CTRL+SPACE` when the curser is right behind the instance variable to get a list of valid members of an instance.

Example:

```
>>> import oser

>>> class SubData(oser.ByteStruct):
...     def __init__(self, *args, **kwargs):
...         super(SubData, self).__init__(*args, **kwargs)
...
...         self.subdata = oser.UBInt16(23)
...
>>> class Data(oser.ByteStruct):
...     def __init__(self):
...         super(Data, self).__init__()
...         self.enum = oser.Enum(prototype=oser.UBInt16,
...                               values={
...                                   "A": 1,
...                                   "B": 2,
...                                   "C": 3,
...                                   "D": 4,
...                               }, value="D")
...
...         self.s = SubData()
...
>>> data = Data()

>>> print(data.s.subdata.get())
23
>>> data.s.subdata.set(3)
>>> print(data.s.subdata.get())
3
>>> print(data.enum.get())
D
```

Reading Values

Every `oser.ByteType` is an instance of a class. Simply use `oser.ByteType.get()` to get the current value.

Since arithmetic emulation is implemented for all primitive types, there is no need to call `oser.ByteType.get()` if you want to compare two values, etc..

Example:

```
>>> from oser import ByteStruct, Enum, UBInt16

>>> class SubData(ByteStruct):
```

```
...     def __init__(self, *args, **kwargs):
...         ByteStruct.__init__(self, *args, **kwargs)
...
...         self.subdata = UBInt16(23)
...
>>> class Data(ByteStruct):
...     def __init__(self):
...         super(Data, self).__init__()
...         self.s = SubData()
...
>>> data = Data()
>>> print(data.s.subdata.get())
23
>>> data.s.subdata.set(3)
>>> print(data.s.subdata.get())
3
```

Setting Values

Every `oser.ByteType` is an instance of a class. Simply use `oser.ByteType.set()` to set the current value.

Example:

```
>>> from oser import ByteStruct, Enum, UBInt16, to_hex

>>> class Data(ByteStruct):
...         def __init__(self):
...             super(Data, self).__init__()
...             self.enum = Enum(prototype=UBInt16,
...                              values={
...                                  "A": 1,
...                                  "B": 2,
...                                  "C": 3,
...                                  "D": 4,
...                              }, value="C")
...
>>> instance = Data()
>>> print(instance)
Data():
    enum: 'C' (UBInt16)

>>> print(instance.introspect())
   -    -  Data():
   0 \x00      enum: 3 (UBInt16)
   1 \x03

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1
\x00\x03
>>> instance.enum.set("B")
>>> print(instance)
Data():
    enum: 'B' (UBInt16)

>>> print(instance.introspect())
```

```
   -    -  Data():
   0 \x00      enum: 2 (UBInt16)
   1 \x02

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1
\x00\x02
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
2
>>> print(instance)
Data():
    enum: 'B' (UBInt16)
```

The Context

Every building block is aware of its context. Conditional building blocks use the context to decide how to proceed.

Variable Length String

The simplest example is a `oser.String` with a variable length.

```
>>> from oser import ByteStruct, String, Switch, Null, IfElse, \
    UBInt8, UBInt16, UBInt32, Array

>>> class VariableLengthString(ByteStruct):
...     def __init__(self):
...         super(VariableLengthString, self).__init__()
...         self.length = UBInt16(1)
...         self.data = String(
...             length=lambda self: self.length.get(),
...             value=b"abcdefghijklmnopqrstuvwxyz")
...
>>> instance = VariableLengthString()
>>> print(instance.introspect())
   -    -  Data():
   0 \x00      length: 1 (UBInt16)
   1 \x01
   -    -      data: String():
   2 \x61          'a'

>>> instance.length.set(16)
>>> print(instance.introspect())
   -    -  Data():
   0 \x00      length: 16 (UBInt16)
   1 \x10
   -    -      data: String():
   2 \x61          'a'
   3 \x62          'b'
   4 \x63          'c'
   5 \x64          'd'
   6 \x65          'e'
   7 \x66          'f'
   8 \x67          'g'
```

```
   9 \x68          'h'
  10 \x69          'i'
  11 \x6a          'j'
  12 \x6b          'k'
  13 \x6c          'l'
  14 \x6d          'm'
  15 \x6e          'n'
  16 \x6f          'o'
  17 \x70          'p'
```

The `length` of the `oser.String` is a `callable` here that returns the `length`'s value.

For simple true-false-decisions `oser.IfElse` can be used.

```
>>> class IfElseData(ByteStruct):
...     def __init__(self):
...         super(IfElseData, self).__init__()
...         self.true_false = UBInt8(1)
...         self.data = IfElse(condition=lambda self: bool(self.true_false.get()),
...                            if_true=UBInt8(1),
...                            if_false=UBInt32(0xffffffff)
...                            )
...
>>> instance = IfElseData()
>>> print(instance.introspect())
   -    -  IfElseData():
   0 \x01      true_false: 1 (UBInt8)
   1 \x01      data: 1 (UBInt8)

>>> instance.true_false.set(0)
>>> print(instance.introspect())
   -    -  IfElseData():
   0 \x00      true_false: 0 (UBInt8)
   1 \xff      data: 4294967295 (UBInt32)
   2 \xff
   3 \xff
   4 \xff
```

The `condition` of `oser.IfElse` is a `callable` here that returns the `true_false`'s value as `bool`.

A more complex example is a `oser.Switch` that decides the type of the payload.

```
>>> class SwitchData(ByteStruct):
...     def __init__(self):
...         super(SwitchData, self).__init__()
...         self.type = UBInt8(1)
...         self.data = Switch(condition=lambda self: self.type.get(),
...                            values={
...                                1: UBInt8(1),
...                                2: UBInt16(2),
...                                4: UBInt32(3)
...                            },
...                            default=Null())
...
```

```
>>> instance = SwitchData()
>>> print(instance.introspect())
   -    -  SwitchData():
   0 \x01      type: 1 (UBInt8)
   1 \x01      data: 1 (UBInt8)

>>> instance.type.set(2)
>>> print(instance.introspect())
   -    -  SwitchData():
   0 \x02      type: 2 (UBInt8)
   1 \x00      data: 2 (UBInt16)
   2 \x02

>>> instance.type.set(4)
>>> print(instance.introspect())
   -    -  SwitchData():
   0 \x04      type: 4 (UBInt8)
   1 \x00      data: 3 (UBInt32)
   2 \x00
   3 \x00
   4 \x03

>>> instance.type.set(3)
>>> print(instance.introspect())
   -    -  SwitchData():
   0 \x03      type: 3 (UBInt8)
   -    -      data: Null
```

The `condition` of `oser.Switch` is a `callable` here that returns the `type`'s value. If `type` is not in `[1,2,4]` `oser.Null` is used.

To access parent elements use `oser.OserNode.up()`.

```
>>> class SubSwitch(ByteStruct):
...     def __init__(self):
...         super(SubSwitch, self).__init__()
...         self.data = Switch(condition=lambda self: self.up().type.get(),
...                            values={
...                                1: UBInt8(1),
...                                2: UBInt16(2),
...                                4: UBInt32(3)
...                            },
...                            default=Null())
...

>>> class SwitchData2(ByteStruct):
...     def __init__(self):
...         super(SwitchData2, self).__init__()
...         self.type = UBInt8(1)
...         self.data = SubSwitch()
...
>>> instance = SwitchData()
>>> print(instance.introspect())
   -    -  SwitchData():
   0 \x01      type: 1 (UBInt8)
   1 \x01      data: 1 (UBInt8)
```

```
>>> instance.type.set(2)
>>> print(instance.introspect())
   -    -  SwitchData():
   0 \x02      type: 2 (UBInt8)
   1 \x00      data: 2 (UBInt16)
   2 \x02

>>> instance.type.set(4)
>>> print(instance.introspect())
   -    -  SwitchData():
   0 \x04      type: 4 (UBInt8)
   1 \x00      data: 3 (UBInt32)
   2 \x00
   3 \x00
   4 \x03

>>> instance.type.set(3)
>>> print(instance.introspect())
   -    -  SwitchData():
   0 \x03      type: 3 (UBInt8)
   -    -      data: Null
```

In this example the `type` is accessed by `self.up().type.get()`.

It is also possible to create variable length repeated fields using `oser.Array`.

```
>>> class VariableLengthArray(ByteStruct):
...     def __init__(self):
...         super(VariableLengthArray, self).__init__()
...         self.length = UBInt16(1)
...         self.data = Array(
...             length=lambda self: self.length.get(),
...             prototype=UBInt8,
...             values=[UBInt8(ii) for ii in range(256)])
...
>>> instance = VariableLengthArray()
>>> print(instance.introspect())
   -    -  VariableLengthArray():
   0 \x00      length: 1 (UBInt16)
   1 \x01
   -    -      data: Array():
   -    -      [
   2 \x00          @0: 0 (UBInt8)
   -    -      ]

>>> instance.length.set(5)
>>> print(instance.introspect())
   -    -  VariableLengthArray():
   0 \x00      length: 5 (UBInt16)
   1 \x05
   -    -      data: Array():
   -    -      [
   2 \x00          @0: 0 (UBInt8)
   3 \x01          @1: 1 (UBInt8)
   4 \x02          @2: 2 (UBInt8)
   5 \x03          @3: 3 (UBInt8)
```

```
    6 \x04          @4: 4 (UBInt8)
    -    -      ]
```

In this example `length` is a `callable` like in the variable length string example.

Copying

OSER instances can be deeply copied using `copy.deepcopy()`. Shallow copies are not allowed since the context of each element must be distinct. If `copy.copy()` is applied on an OSER instance and exception is raised. While being copied `encode()` is called with all its side effects.

```
>>> import oser
>>> import copy


>>> class Struct(oser.ByteStruct):
...     def __init__(self):
...         super(Struct, self).__init__()
...         self.a = oser.ULInt8(1)
...
>>> s = Struct()
>>> id(s)
6596088
>>> print(s)
Struct():
    a: 1 (ULInt8)


>>> deep_copy = copy.deepcopy(s)
>>> id(deep_copy)
38738520

>>> s.a.set(100)  # this does not influence the copy
>>> print(deep_copy)
Struct():
    a: 1 (ULInt8)
```

Examples

EEPROM-Content

Let's say you want to create an eeprom content using OSER.

The content has the following structure:

Entry	Type	Position	Description
type	U8	0	the module type
serial number	String[9]	1 .. 9	serial number: four upper case letters, a minus and four digits
hardware-type	U8	10	the hardware-type

Entry	Type	Position	Description
calibration	F32[2]	11 .. 18	calibration factors
crc	U32	12 .. 22	32-bit CRC, polynomial 0x1EDC6F41

All values are stored in big endian.

The following abstraction lets you create this mapping easily:

```
>>> from oser import ByteStruct, UBInt8, BFloat, RegularExpressionMatch, \
    Array, CRCB32, to_hex

>>> class EEPROM(ByteStruct):
...     def __init__(self, *args, **kwargs):
...         ByteStruct.__init__(self, *args, **kwargs)
...
...         self.type = UBInt8(0)
...         self.serial_number = RegularExpressionMatch(pattern=r"[A-Z]{4}-[0-9]{4}", length=9,
value=b"AAAA-0000")
...         self.hardware_type = UBInt8(0)
...         self.calibration = Array(length=2, prototype=BFloat)
...         self.crc = CRCB32(strict=True, polynomial=0x1EDC6F41)
...
>>> eeprom = EEPROM()
>>> eeprom.type.set(3)
>>> eeprom.serial_number.set(b"ASDF-1337")
>>> eeprom.hardware_type.set(23)
>>> for ii in range(2):
...     eeprom.calibration[ii].set((.1+ii)**3)
...
>>> binary = eeprom.encode()
>>> 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
\x03\x41\x53\x44\x46\x2D\x31\x33\x33\x37\x17\x3A\x83\x12\x6F\x3F\xAA\x5E\x35\x3D\x10\xBB\xCB
>>> print(eeprom.introspect())
   -    -  EEPROM():
   0 \x03      type: 3 (UBInt8)
   -    -      serial_number: RegularExpressionMatch():
   1 \x41          'A'
   2 \x53          'S'
   3 \x44          'D'
   4 \x46          'F'
   5 \x2d          '-'
   6 \x31          '1'
   7 \x33          '3'
   8 \x33          '3'
   9 \x37          '7'
  10 \x17      hardware_type: 23 (UBInt8)
   -    -      calibration: Array():
   -    -      [
  11 \x3a          @0: 0.001 (BFloat)
  12 \x83
  13 \x12
  14 \x6f
  15 \x3f          @1: 1.331 (BFloat)
  16 \xaa
  17 \x5e
  18 \x35
   -    -      ]
```

```
  19 \x3d      crc: 1024506827 (CRCB32)
  20 \x10
  21 \xbb
  22 \xcb
```

You can now write this data into your eeprom.

Custom Network Protocol

You may want to create a custom network protocol for you device or software. Because the message is transmitted via any serial connection it starts with a magic number `0x23`. This is a `U8`.

This magic number is followed by the destination address and the source address. Both are `U16`.

Additionally there is a service-id as `U32`.

The service-id is followed by the total payload length as `U16` without crc. This way the transport-layer must not be aware of the protocol itself.

The payload consists of a type, a length and the data itself. The type has symbolic names:

- 0: string
- 1: list of 16-bit integers
- 2: list of floats

A frame ends with a 32-bit crc with polynomial 0x1EDC6F41.

All values are transmitted in big endian.

The following abstraction creates a frame for this protocol:

```
>>> from oser import ByteStruct, Constant, UBInt8, BFloat, UBInt16, UBInt32, \
...     Enum, Array, Switch, String, CRCB32, to_hex
>>>
>>> a1 = Array(length=1, prototype=UBInt16)
>>> a2 = Array(length=2, prototype=BFloat)
>>>
>>> class SerialProtocol(ByteStruct):
...     def __init__(self, *args, **kwargs):
...         ByteStruct.__init__(self, *args, **kwargs)
...
...         self.start_of_frame = Constant(UBInt8, 0x23)
...         self.destination = UBInt16(0)
...         self.source = UBInt16(0)
...         self.service_id = UBInt32(0)
...         self.payload_length = UBInt16(0)
...         self.payload = Payload()
...         self.crc = CRCB32(strict=True, polynomial=0x1EDC6F41)
...
...     def encode(self, full_data=b"", context_data=b""):
...         self.payload_length.set(self.payload.size())
...         return super(SerialProtocol, self).encode(full_data, context_data)
...
>>> class Payload(ByteStruct):
```

```
...     def __init__(self, *args, **kwargs):
...         ByteStruct.__init__(self, *args, **kwargs)
...
...         self.type = Enum(UBInt32,
...                          values={
...                              "string": 0,
...                              "u16list": 1,
...                              "f32list": 2,
...                          }, value="string")
...         self.length = UBInt16(0)
...         self.data = Switch(
...             condition=lambda self: self.type.get(),
...             values={
...                 "string": String(length=lambda self: self.length.get()),
...                 "u16list": Array(length=lambda self: self.length.get(), prototype=UBInt16),
...                 "f32list": Array(length=lambda self: self.length.get(), prototype=BFloat),
...             })
...
>>> frame = SerialProtocol()
>>> frame.service_id.set(1337)
>>>
>>> # string
>>> frame.payload.type.set("string")
>>> frame.payload.length.set(10)
>>> frame.payload.data.set("abcdefghij")
>>>
>>> binary = frame.encode()
>>> print(frame)
SerialProtocol():
    start_of_frame: 35 (UBInt8)
    destination: 0 (UBInt16)
    source: 0 (UBInt16)
    service_id: 1337 (UBInt32)
    payload_length: 16 (UBInt16)
    payload: Payload():
        type: 'string' (UBInt32)
        length: 10 (UBInt16)
        data: b'abcdefghij'
    crc: 0x57c9544f (CRCB32)

>>> 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| 28| 29| 30
\x23\x00\x00\x00\x00\x00\x05\x39\x00\x10\x00\x00\x00\x00\x00\x0a\x61\x62\x63\x64\x65\x66\x67\x68
\x69\x6a\x57\xc9\x54\x4f
>>> print(frame.introspect())
   -    -  SerialProtocol():
   0 \x23      start_of_frame: 35 (UBInt8)
   1 \x00      destination: 0 (UBInt16)
   2 \x00
   3 \x00      source: 0 (UBInt16)
   4 \x00
   5 \x00      service_id: 1337 (UBInt32)
   6 \x00
   7 \x05
   8 \x39
   9 \x00      payload_length: 16 (UBInt16)
  10 \x10
   -    -      payload: Payload():
  11 \x00          type: 0 (UBInt32)
```

```
  12 \x00
  13 \x00
  14 \x00
  15 \x00          length: 10 (UBInt16)
  16 \x0a
   -    -          data: String():
  17 \x61            'a'
  18 \x62            'b'
  19 \x63            'c'
  20 \x64            'd'
  21 \x65            'e'
  22 \x66            'f'
  23 \x67            'g'
  24 \x68            'h'
  25 \x69            'i'
  26 \x6a            'j'
  27 \x57      crc: 0x57c9544f (CRCB32)
  28 \xc9
  29 \x54
  30 \x4f

>>> # list of u16
>>> frame.payload.type.set("u16list")
>>> frame.payload.length.set(3)
>>> for ii in range(3):
...     frame.payload.data[ii].set(ii)
...
>>> binary = frame.encode()
>>> print(frame)
SerialProtocol():
    start_of_frame: 35 (UBInt8)
    destination: 0 (UBInt16)
    source: 0 (UBInt16)
    service_id: 1337 (UBInt32)
    payload_length: 12 (UBInt16)
    payload: Payload():
        type: 'u16list' (UBInt32)
        length: 3 (UBInt16)
        data: Array():
        [
            @0: 0 (UBInt16)
            @1: 1 (UBInt16)
            @2: 2 (UBInt16)
        ]
    crc: 0x224b28ee (CRCB32)

>>> 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
\x23\x00\x00\x00\x00\x00\x05\x39\x00\x0c\x00\x00\x00\x01\x00\x03\x00\x00\x00\x01\x00\x02\x22\x4b
\x28\xee
>>> print(frame.introspect())
   -    -  SerialProtocol():
   0 \x23      start_of_frame: 35 (UBInt8)
   1 \x00      destination: 0 (UBInt16)
   2 \x00
   3 \x00      source: 0 (UBInt16)
   4 \x00
   5 \x00      service_id: 1337 (UBInt32)
   6 \x00
```

```
   7 \x05
   8 \x39
   9 \x00      payload_length: 12 (UBInt16)
  10 \x0c
   -    -      payload: Payload():
  11 \x00          type: 1 (UBInt32)
  12 \x00
  13 \x00
  14 \x01
  15 \x00          length: 3 (UBInt16)
  16 \x03
   -    -          data: Array():
   -    -          [
  17 \x00              @0: 0 (UBInt16)
  18 \x00
  19 \x00              @1: 1 (UBInt16)
  20 \x01
  21 \x00              @2: 2 (UBInt16)
  22 \x02
   -    -          ]
  23 \x22      crc: 0x224b28ee (CRCB32)
  24 \x4b
  25 \x28
  26 \xee

>>> # list of float
>>> frame.payload.type.set("f32list")
>>> frame.payload.length.set(2)
>>> for ii in range(2):
...     frame.payload.data[ii].set(float(ii*1.5))
...
>>> binary = frame.encode()
>>> print(frame)
SerialProtocol():
    start_of_frame: 35 (UBInt8)
    destination: 0 (UBInt16)
    source: 0 (UBInt16)
    service_id: 1337 (UBInt32)
    payload_length: 14 (UBInt16)
    payload: Payload():
        type: 'f32list' (UBInt32)
        length: 2 (UBInt16)
        data: Array():
        [
            @0: 0.0 (BFloat)
            @1: 1.5 (BFloat)
        ]
    crc: 0x4db8bf4 (CRCB32)

>>> 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| 28
\x23\x00\x00\x00\x00\x00\x05\x39\x00\x0e\x00\x00\x00\x02\x00\x02\x00\x00\x00\x00\x3f\xc0\x00\x00
\x04\xdb\x8b\xf4
>>> print(frame.introspect())
   -    -  SerialProtocol():
   0 \x23      start_of_frame: 35 (UBInt8)
   1 \x00      destination: 0 (UBInt16)
   2 \x00
   3 \x00      source: 0 (UBInt16)
```

```
   4 \x00
   5 \x00      service_id: 1337 (UBInt32)
   6 \x00
   7 \x05
   8 \x39
   9 \x00      payload_length: 14 (UBInt16)
  10 \x0e
   -    -      payload: Payload():
  11 \x00          type: 2 (UBInt32)
  12 \x00
  13 \x00
  14 \x02
  15 \x00          length: 2 (UBInt16)
  16 \x02
   -    -          data: Array():
   -    -          [
  17 \x00              @0: 0.0 (BFloat)
  18 \x00
  19 \x00
  20 \x00
  21 \x3f              @1: 1.5 (BFloat)
  22 \xc0
  23 \x00
  24 \x00
   -    -          ]
  25 \x04      crc: 0x4db8bf4 (CRCB32)
  26 \xdb
  27 \x8b
  28 \xf4

>>> #decode data
>>> data = b"\x23\x00\x00\x00\x00\x00\x05\x39\x00\x1A\x00\x00\x00\x02\x00" \
...        b"\x05\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00" \
...        b"\x00\x00\x00\x00\x00\xA4\x58\x03\x64"
>>> frame.decode(data)
41
>>> print(frame)
SerialProtocol():
    start_of_frame: 35 (UBInt8)
    destination: 0 (UBInt16)
    source: 0 (UBInt16)
    service_id: 1337 (UBInt32)
    payload_length: 26 (UBInt16)
    payload: Payload():
        type: 'f32list' (UBInt32)
        length: 5 (UBInt16)
        data: Array():
        [
            @0: 0.0 (BFloat)
            @1: 0.0 (BFloat)
            @2: 0.0 (BFloat)
            @3: 0.0 (BFloat)
            @4: 0.0 (BFloat)
        ]
    crc: 0xa4580364 (CRCB32)

>>> print(frame.introspect())
   -    -  SerialProtocol():
   0 \x23      start_of_frame: 35 (UBInt8)
   1 \x00      destination: 0 (UBInt16)
```

```
 2 \x00
 3 \x00      source: 0 (UBInt16)
 4 \x00
 5 \x00      service_id: 1337 (UBInt32)
 6 \x00
 7 \x05
 8 \x39
 9 \x00      payload_length: 26 (UBInt16)
10 \x1a
 -    -      payload: Payload():
11 \x00          type: 2 (UBInt32)
12 \x00
13 \x00
14 \x02
15 \x00          length: 5 (UBInt16)
16 \x05
 -    -          data: Array():
 -    -          [
17 \x00              @0: 0.0 (BFloat)
18 \x00
19 \x00
20 \x00
21 \x00              @1: 0.0 (BFloat)
22 \x00
23 \x00
24 \x00
25 \x00              @2: 0.0 (BFloat)
26 \x00
27 \x00
28 \x00
29 \x00              @3: 0.0 (BFloat)
30 \x00
31 \x00
32 \x00
33 \x00              @4: 0.0 (BFloat)
34 \x00
35 \x00
36 \x00
 -    -          ]
37 \xa4      crc: 0xa4580364 (CRCB32)
38 \x58
39 \x03
40 \x64
```

HTTP-Request

This example shows how to abstract a http request transmitted via TCP and IPv4.

First an ethernet frame must be constructed:

```
class EthernetFrame(ByteStruct):
    def __init__(self, *args, **kwargs):
        ByteStruct.__init__(self, *args, **kwargs)

        self.destination = MacAddress(6)
        self.source = MacAddress(6)
```

```
        self.type = Enum(prototype=UBInt16, values={"IP" : 0x0800})

        self.payload = Switch(condition=lambda self: self.type.get(),
                              values={
                                  "IP" : InternetProtocolPayload(),
                              },
                              default=Null())
```

The MAC address is a `oser.String` with little modifications for printing:

```
class MacAddress(String):
    def get(self):
        return ":".join(map(lambda c: f"{c:X}", super(MacAddress, self).get()))
```

Now the IPv4 payload must be constructed:

```
class InternetProtocolPayload(ByteStruct):
    def __init__(self, *args, **kwargs):
        ByteStruct.__init__(self, *args, **kwargs)

        self.header = BitStruct()
        self.header.version = Nibble(0)
        self.header.ihl = Nibble(0)  # ip header type
        self.header.tos = BitStruct()  # type of service
        self.header.dscp = BitField(length=6)  # differentiated service code point
        self.header.ecn = BitField(length=2)  # explicit congestion notification (ip flowcontrol)
        self.header.total_length = BitField(length=16)
        self.header.identification = BitField(length=16)
        self.header.flags = BitField(length=3)
        self.header.fragment_offset = BitField(length=13)
        self.header.ttl = BitField(length=8)
        self.protocol = Enum(UBInt8, values={"TCP" : 0x06})
        self.header_checksum = UBInt16(0)

        self.payload = Switch(condition=lambda self: self.protocol.get(),
                              values={
                                  "TCP" : TransmissionControlProtocolPayload(),
                              })
```

And for TCP a payload must be constructed, too:

```
class TransmissionControlProtocolPayload(ByteStruct):
    def __init__(self, *args, **kwargs):
        ByteStruct.__init__(self, *args, **kwargs)

        self.source_address = UBInt32(0)
        self.destination_address = UBInt32(0)
        self.source_port = UBInt16(0)
        self.destination_port = UBInt16(0)
        self.sequence_number = UBInt32(0)
        self.acknowledgement_number = UBInt32(0)
```

```
        self.offset_and_flags = BitStruct()
        self.offset_and_flags.offset = BitField(length=4)
        self.offset_and_flags.reserved = BitField(length=6)
        self.offset_and_flags.urgent_flag = Flag()
        self.offset_and_flags.acknowledgement_flag = Flag()
        self.offset_and_flags.psh_flag = Flag()
        self.offset_and_flags.rst_flag = Flag()
        self.offset_and_flags.syn_flag = Flag()
        self.offset_and_flags.fin_flag = Flag()

        self.window = UBInt16()
        self.checksum = UBInt16()
        self.urgent_pointer = UBInt16()

        self.options = Array(length=3, prototype=UBInt32)

        self.payload = HTTPPayload(length=lambda self: self.up().header.total_length.get()-52)
```

The HTTP payload is a `oser.String` with little modifications too. That helps to show the plain text payload in a human readable way:

```
class HTTPPayload(String):
    def _toString(self):
        return "\n" + "\n".join(self.get().split("\r\n"))
```

To decode an http request use the following code (the data came from Wireshark. Right-click a packet -> Copy -> Bytes -> Hex Stream):

```
>>> frame = EthernetFrame()

>>> data = binascii.unhexlify(
...     b"f4ec38c9b85438607713fa070800450003162ac740004006fabbc0a80003c163"
...     b"9050ac2f005062b7938d1e3de4bb80187210156800000101080a0173bd0013d6"
...     b"d56f474554202f20485454502f312e310d0a486f73743a2068656973652e6465"
...     b"0d0a436f6e6e656374696f6e3a206b6565702d616c6976650d0a557365722d41"
...     b"67656e743a204d6f7a696c6c612f352e3020285831313b204c696e7578207838"
...     b"365f363429204170706c655765624b69742f3533352e313920284b48544d4c2c"
...     b"206c696b65204765636b6f29204368726f6d652f31382e302e313032352e3136"
...     b"32205361666172692f3533352e31390d0a4163636570743a20746578742f6874"
...     b"6d6c2c6170706c69636174696f6e2f7868746d6c2b786d6c2c6170706c696361"
...     b"74696f6e2f786d6c3b713d302e392c2a2f2a3b713d302e380d0a416363657074"
...     b"2d456e636f64696e673a20677a69702c6465666c6174652c736463680d0a4163"
...     b"636570742d4c616e67756167653a2064652d44452c64653b713d302e382c656e"
...     b"2d55533b713d302e362c656e3b713d302e340d0a4163636570742d4368617273"
...     b"65743a2049534f2d383835392d312c7574662d383b713d302e372c2a3b713d30"
...     b"2e330d0a436f6f6b69653a2077745f76743d3776303030253342367633357431"
...     b"3430393539373433353731383b207774335f6569643d25334232383836383936"
...     b"3336393230313734253743323134323531313732373230303232303339313b20"
...     b"6f7074696d697a656c79456e645573657249643d6f6575313432383735373231"
...     b"3735343972302e393837303834313034393132333535353b206f7074696d697a"
...     b"656c795365676d656e74733d25374225323232323836323530383538253232 25"
...     b"3341253232676325323225324325323232333031313130373332253232253341"
...     b"25323266616c7365253232253243253232323331363235303236342532322533"
...     b"4125323273656172636825323225374 3b206f7074696d697a656c794275636b"
```

```
...     b"6574733d25374225323232373435383230343039253232253341253232302532"
...     b"322537443b207774335f7369643d25334232383836383936333639323031373 4"
...     b"0d0a0d0a"
... )

>>> frame.decode(data)
804
>>> print(frame)
EthernetFrame():
    destination: 'F4:EC:38:C9:B8:54'
    source: '38:60:77:13:FA:7'
    type: 'IP' (UBInt16)
    payload: InternetProtocolPayload():
        header: BitStruct():
            version: 4 (Nibble)
            ihl: 5 (Nibble)
            tos: BitStruct():
            dscp: 0 (BitField(6))
            ecn: 0 (BitField(2))
            total_length: 790 (BitField(16))
            identification: 10951 (BitField(16))
            flags: 2 (BitField(3))
            fragment_offset: 0 (BitField(13))
            ttl: 64 (BitField(8))
        protocol: 'TCP' (UBInt8)
        header_checksum: 64187 (UBInt16)
        payload: TransmissionControlProtocolPayload():
            source_address: 3232235523 (UBInt32)
            destination_address: 3244527696 (UBInt32)
            source_port: 44079 (UBInt16)
            destination_port: 80 (UBInt16)
            sequence_number: 1656198029 (UBInt32)
            acknowledgement_number: 507372731 (UBInt32)
            offset_and_flags: BitStruct():
                offset: 8 (BitField(4))
                reserved: 0 (BitField(6))
                urgent_flag: False (Flag)
                acknowledgement_flag: True (Flag)
                psh_flag: True (Flag)
                rst_flag: False (Flag)
                syn_flag: False (Flag)
                fin_flag: False (Flag)
            window: 29200 (UBInt16)
            checksum: 5480 (UBInt16)
            urgent_pointer: 0 (UBInt16)
            options: Array():
            [
                @0: 16844810 (UBInt32)
                @1: 24362240 (UBInt32)
                @2: 332846447 (UBInt32)
            ]
            payload: b'GET / HTTP/1.1\r\nHost: heise.de\r\nConnection: keep-alive\r\nUser-Agent:
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/535.19 (KHTML, like Gecko) Chrome/18.0.1025.162 Safari/
535.19\r\nAccept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\nAccept-
Encoding: gzip,deflate,sdch\r\nAccept-Language: de-DE,de;q=0.8,en-US;q=0.6,en;q=0.4\r\nAccept-
Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3\r\nCookie: wt_vt=7v000%3B6v35t1409597435718;
wt3_eid=%3B288689636920174%7C2142511727200220391;
optimizelyEndUserId=oeu1428757217549r0.9870841049123555;
optimizelySegments=%7B%222286250858%22%3A%22gc%22%2C%222301110732%22%3A%22false%22%2C%222316250264%2
2%3A%22search%22%7D; optimizelyBuckets=%7B%222745820409%22%3A%220%22%7D;
```

```
wt3_sid=%3B288689636920174\r\n\r\n'

>>> print(frame.introspect())
   -    -  EthernetFrame():
   -    -      destination: MacAddress():
   0 \xf4        'ô'
   1 \xec        'ì'
   2 \x38        '8'
   3 \xc9        'É'
   4 \xb8        '¸'
   5 \x54        'T'
   -    -      source: MacAddress():
   6 \x38        '8'
   7 \x60        '`'
   8 \x77        'w'
   9 \x13        '\x13'
  10 \xfa        'ú'
  11 \x07        '\x07'
  12 \x08      type: 2048 (UBInt16)
  13 \x00
   -    -      payload: InternetProtocolPayload():
   -    -          header: BitStruct():
  14.7  0             version: 4 (Nibble)
  14.6  1
  14.5  0
  14.4  0
  14.3  0             ihl: 5 (Nibble)
  14.2  1
  14.1  0
  14.0  1
   -    -             tos: BitStruct():
  15.7  0             dscp: 0 (BitField(6))
  15.6  0
  15.5  0
  15.4  0
  15.3  0
  15.2  0
  15.1  0             ecn: 0 (BitField(2))
  15.0  0
  16.7  0             total_length: 790 (BitField(16))
  16.6  0
  16.5  0
  16.4  0
  16.3  0
  16.2  0
  16.1  1
  16.0  1
  17.7  0
  17.6  0
  17.5  0
  17.4  1
  17.3  0
  17.2  1
  17.1  1
  17.0  0
  18.7  0             identification: 10951 (BitField(16))
  18.6  0
  18.5  1
  18.4  0
  18.3  1
```

```
18.2  0
18.1  1
18.0  0
19.7  1
19.6  1
19.5  0
19.4  0
19.3  0
19.2  1
19.1  1
19.0  1
20.7  0             flags: 2 (BitField(3))
20.6  1
20.5  0
20.4  0             fragment_offset: 0 (BitField(13))
20.3  0
20.2  0
20.1  0
20.0  0
21.7  0
21.6  0
21.5  0
21.4  0
21.3  0
21.2  0
21.1  0
21.0  0
22.7  0             ttl: 64 (BitField(8))
22.6  1
22.5  0
22.4  0
22.3  0
22.2  0
22.1  0
22.0  0
23 \x06          protocol: 6 (UBInt8)
24 \xfa          header_checksum: 64187 (UBInt16)
25 \xbb
 -    -          payload: TransmissionControlProtocolPayload():
26 \xc0              source_address: 3232235523 (UBInt32)
27 \xa8
28 \x00
29 \x03
30 \xc1              destination_address: 3244527696 (UBInt32)
31 \x63
32 \x90
33 \x50
34 \xac              source_port: 44079 (UBInt16)
35 \x2f
36 \x00              destination_port: 80 (UBInt16)
37 \x50
38 \x62              sequence_number: 1656198029 (UBInt32)
39 \xb7
40 \x93
41 \x8d
42 \x1e              acknowledgement_number: 507372731 (UBInt32)
43 \x3d
44 \xe4
45 \xbb
 -    -              offset_and_flags: BitStruct():
```

```
  46.7  1                 offset: 8 (BitField(4))
  46.6  0
  46.5  0
  46.4  0
  46.3  0                 reserved: 0 (BitField(6))
  46.2  0
  46.1  0
  46.0  0
  47.7  0
  47.6  0
  47.5  0                 urgent_flag: False (Flag)
  47.4  1                 acknowledgement_flag: True (Flag)
  47.3  1                 psh_flag: True (Flag)
  47.2  0                 rst_flag: False (Flag)
  47.1  0                 syn_flag: False (Flag)
  47.0  0                 fin_flag: False (Flag)
  48 \x72             window: 29200 (UBInt16)
  49 \x10
  50 \x15             checksum: 5480 (UBInt16)
  51 \x68
  52 \x00             urgent_pointer: 0 (UBInt16)
  53 \x00
   -    -             options: Array():
   -    -             [
  54 \x01                 @0: 16844810 (UBInt32)
  55 \x01
  56 \x08
  57 \x0a
  58 \x01                 @1: 24362240 (UBInt32)
  59 \x73
  60 \xbd
  61 \x00
  62 \x13                 @2: 332846447 (UBInt32)
  63 \xd6
  64 \xd5
  65 \x6f
   -    -             ]
   -    -             payload: HTTPPayload():
  66 \x47               'G'
  67 \x45               'E'
  68 \x54               'T'
  69 \x20               ' '
  70 \x2f               '/'
  71 \x20               ' '
  72 \x48               'H'
  73 \x54               'T'
  74 \x54               'T'
  75 \x50               'P'
  76 \x2f               '/'
  77 \x31               '1'
  78 \x2e               '.'
  79 \x31               '1'
 %< %< %< %< %< %< %<
 800 \x0d               '\r'
 801 \x0a               '\n'
 802 \x0d               '\r'
 803 \x0a               '\n'

>>> binary = frame.encode()
>>> 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| 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| 86| 87| 88| 89| 90| 91| 92| 93| 94| 95| 96| 97| 98|
 99|100|101|102|103|104|105|106|107|108|109|110|111|112|113|114|115|116|117|118|119|120|121|122|123|
124|125|126|127|128|129|130|131|132|133|134|135|136|137|138|139|140|141|142|143|144|145|146|147|148|
149|150|151|152|153|154|155|156|157|158|159|160|161|162|163|164|165|166|167|168|169|170|171|172|173|
174|175|176|177|178|179|180|181|182|183|184|185|186|187|188|189|190|191|192|193|194|195|196|197|198|
199|200|201|202|203|204|205|206|207|208|209|210|211|212|213|214|215|216|217|218|219|220|221|222|223|
224|225|226|227|228|229|230|231|232|233|234|235|236|237|238|239|240|241|242|243|244|245|246|247|248|
249|250|251|252|253|254|255|256|257|258|259|260|261|262|263|264|265|266|267|268|269|270|271|272|273|
274|275|276|277|278|279|280|281|282|283|284|285|286|287|288|289|290|291|292|293|294|295|296|297|298|
299|300|301|302|303|304|305|306|307|308|309|310|311|312|313|314|315|316|317|318|319|320|321|322|323|
324|325|326|327|328|329|330|331|332|333|334|335|336|337|338|339|340|341|342|343|344|345|346|347|348|
349|350|351|352|353|354|355|356|357|358|359|360|361|362|363|364|365|366|367|368|369|370|371|372|373|
374|375|376|377|378|379|380|381|382|383|384|385|386|387|388|389|390|391|392|393|394|395|396|397|398|
399|400|401|402|403|404|405|406|407|408|409|410|411|412|413|414|415|416|417|418|419|420|421|422|423|
424|425|426|427|428|429|430|431|432|433|434|435|436|437|438|439|440|441|442|443|444|445|446|447|448|
449|450|451|452|453|454|455|456|457|458|459|460|461|462|463|464|465|466|467|468|469|470|471|472|473|
474|475|476|477|478|479|480|481|482|483|484|485|486|487|488|489|490|491|492|493|494|495|496|497|498|
499|500|501|502|503|504|505|506|507|508|509|510|511|512|513|514|515|516|517|518|519|520|521|522|523|
524|525|526|527|528|529|530|531|532|533|534|535|536|537|538|539|540|541|542|543|544|545|546|547|548|
549|550|551|552|553|554|555|556|557|558|559|560|561|562|563|564|565|566|567|568|569|570|571|572|573|
574|575|576|577|578|579|580|581|582|583|584|585|586|587|588|589|590|591|592|593|594|595|596|597|598|
599|600|601|602|603|604|605|606|607|608|609|610|611|612|613|614|615|616|617|618|619|620|621|622|623|
624|625|626|627|628|629|630|631|632|633|634|635|636|637|638|639|640|641|642|643|644|645|646|647|648|
649|650|651|652|653|654|655|656|657|658|659|660|661|662|663|664|665|666|667|668|669|670|671|672|673|
674|675|676|677|678|679|680|681|682|683|684|685|686|687|688|689|690|691|692|693|694|695|696|697|698|
699|700|701|702|703|704|705|706|707|708|709|710|711|712|713|714|715|716|717|718|719|720|721|722|723|
724|725|726|727|728|729|730|731|732|733|734|735|736|737|738|739|740|741|742|743|744|745|746|747|748|
749|750|751|752|753|754|755|756|757|758|759|760|761|762|763|764|765|766|767|768|769|770|771|772|773|
774|775|776|777|778|779|780|781|782|783|784|785|786|787|788|789|790|791|792|793|794|795|796|797|798|
799|800|801|802|803
\xf4\xec\x38\xc9\xb8\x54\x38\x60\x77\x13\xfa\x07\x08\x00\x45\x00\x03\x16\x2a\xc7\x40\x00\x40\x06\xfa
\xbb\xc0\xa8\x00\x03\xc1\x63\x90\x50\xac\x2f\x00\x50\x62\xb7\x93\x8d\x1e\x3d\xe4\xbb\x80\x18\x72\x10
\x15\x68\x00\x00\x01\x01\x08\x0a\x01\x73\xbd\x00\x13\xd6\xd5\x6f\x47\x45\x54\x20\x2f\x20\x48\x54\x54
\x50\x2f\x31\x2e\x31\x0d\x0a\x48\x6f\x73\x74\x3a\x20\x68\x65\x69\x73\x65\x2e\x64\x65\x0d\x0a\x43\x6f
\x6e\x6e\x65\x63\x74\x69\x6f\x6e\x3a\x20\x6b\x65\x65\x70\x2d\x61\x6c\x69\x76\x65\x0d\x0a\x55\x73\x65
\x72\x2d\x41\x67\x65\x6e\x74\x3a\x20\x4d\x6f\x7a\x69\x6c\x6c\x61\x2f\x35\x2e\x30\x20\x28\x58\x31\x31
\x3b\x20\x4c\x69\x6e\x75\x78\x20\x78\x38\x36\x5f\x36\x34\x29\x20\x41\x70\x70\x6c\x65\x57\x65\x62\x4b
\x69\x74\x2f\x35\x33\x35\x2e\x31\x39\x20\x28\x4b\x48\x54\x4d\x4c\x2c\x20\x6c\x69\x6b\x65\x20\x47\x65
\x63\x6b\x6f\x29\x20\x43\x68\x72\x6f\x6d\x65\x2f\x31\x38\x2e\x30\x2e\x31\x30\x32\x35\x2e\x31\x36\x32
\x20\x53\x61\x66\x61\x72\x69\x2f\x35\x33\x35\x2e\x31\x39\x0d\x0a\x41\x63\x63\x65\x70\x74\x3a\x20\x74
\x65\x78\x74\x2f\x68\x74\x6d\x6c\x2c\x61\x70\x70\x6c\x69\x63\x61\x74\x69\x6f\x6e\x2f\x78\x68\x74\x6d
\x6c\x2b\x78\x6d\x6c\x2c\x61\x70\x70\x6c\x69\x63\x61\x74\x69\x6f\x6e\x2f\x78\x6d\x6c\x3b\x71\x3d\x30
\x2e\x39\x2c\x2a\x2f\x2a\x3b\x71\x3d\x30\x2e\x38\x0d\x0a\x41\x63\x63\x65\x70\x74\x2d\x45\x6e\x63\x6f
\x64\x69\x6e\x67\x3a\x20\x67\x7a\x69\x70\x2c\x64\x65\x66\x6c\x61\x74\x65\x2c\x73\x64\x63\x68\x0d\x0a
\x41\x63\x63\x65\x70\x74\x2d\x4c\x61\x6e\x67\x75\x61\x67\x65\x3a\x20\x64\x65\x2d\x44\x45\x2c\x64\x65
\x3b\x71\x3d\x30\x2e\x38\x2c\x65\x6e\x2d\x55\x53\x3b\x71\x3d\x30\x2e\x36\x2c\x65\x6e\x3b\x71\x3d\x30
\x2e\x34\x0d\x0a\x41\x63\x63\x65\x70\x74\x2d\x43\x68\x61\x72\x73\x65\x74\x3a\x20\x49\x53\x4f\x2d\x38
\x38\x35\x39\x2d\x31\x2c\x75\x74\x66\x2d\x38\x3b\x71\x3d\x30\x2e\x37\x2c\x2a\x3b\x71\x3d\x30\x2e\x33
\x0d\x0a\x43\x6f\x6f\x6b\x69\x65\x3a\x20\x77\x74\x5f\x76\x74\x3d\x37\x76\x30\x30\x30\x25\x33\x42\x36
\x76\x33\x35\x74\x31\x34\x30\x39\x35\x39\x37\x34\x33\x35\x37\x31\x38\x3b\x20\x77\x74\x33\x5f\x65\x69
\x64\x3d\x25\x33\x42\x32\x38\x38\x36\x38\x39\x36\x33\x36\x39\x32\x30\x31\x37\x34\x25\x37\x43\x32\x31
\x34\x32\x35\x31\x31\x37\x32\x37\x32\x30\x30\x32\x32\x30\x33\x39\x31\x3b\x20\x6f\x70\x74\x69\x6d\x69
\x7a\x65\x6c\x79\x45\x6e\x64\x55\x73\x65\x72\x49\x64\x3d\x6f\x65\x75\x31\x34\x32\x38\x37\x35\x37\x32
\x31\x37\x35\x34\x39\x72\x30\x2e\x39\x38\x37\x30\x38\x34\x31\x30\x34\x39\x31\x32\x33\x35\x35\x35\x3b
\x20\x6f\x70\x74\x69\x6d\x69\x7a\x65\x6c\x79\x53\x65\x67\x6d\x65\x6e\x74\x73\x3d\x25\x37\x42\x25\x32
\x32\x32\x32\x38\x36\x32\x35\x30\x38\x35\x38\x25\x32\x32\x25\x33\x41\x25\x32\x32\x67\x63\x25\x32\x32
\x25\x32\x43\x25\x32\x32\x32\x33\x30\x31\x31\x31\x30\x37\x33\x32\x25\x32\x32\x25\x33\x41\x25\x32\x32
```

```
\x66\x61\x6c\x73\x65\x25\x32\x32\x25\x32\x43\x25\x32\x32\x32\x33\x31\x36\x32\x35\x30\x32\x36\x34\x25
\x32\x32\x25\x33\x41\x25\x32\x32\x73\x65\x61\x72\x63\x68\x25\x32\x32\x25\x37\x44\x3b\x20\x6f\x70\x74
\x69\x6d\x69\x7a\x65\x6c\x79\x42\x75\x63\x6b\x65\x74\x73\x3d\x25\x37\x42\x25\x32\x32\x32\x37\x34\x35
\x38\x32\x30\x34\x30\x39\x25\x32\x32\x25\x33\x41\x25\x32\x32\x30\x25\x32\x32\x25\x37\x44\x3b\x20\x77
\x74\x33\x5f\x73\x69\x64\x3d\x25\x33\x42\x32\x38\x38\x36\x38\x39\x36\x33\x36\x39\x32\x30\x31\x37\x34
\x0d\x0a\x0d\x0a
>>> print(len(binary))
804
```

Access Members By Name

To access members by name call `getattr()` to get the member and call `oser.ByteType.set()` on the result.

```
>>> from oser import ByteStruct, ULInt8

>>> class Foo(ByteStruct):
...     def __init__(self):
...         super(Foo, self).__init__()
...         self.a = ULInt8(1)
...
>>> foo = Foo()
>>> name = "a"
>>> getattr(foo, name).set(24)
>>> print(foo.introspect())
   -    -  Foo():
   0 \x18      a: 24 (ULInt8)

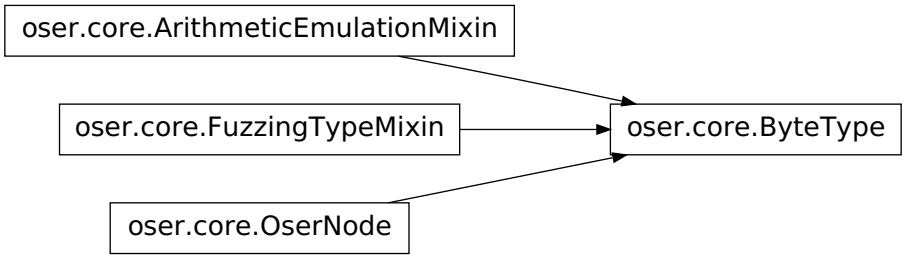
>>> print(foo.a == 24)
True
```

Primitive Types

Primitive types include byte-aligned types like integer numbers, floating point numbers and bit-aligned types like flags, bitfields and also flags.

ByteType

ByteType is an abstract super class which can be used to create byte types.




```
class oser.ByteType(value: Any = 0, format: str | None = None)
```

A `ByteType` is an abstract class for byte types.

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() → Any
```

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: 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 – the values used for fuzzing.

```
up() → ByteStruct | BitStruct
```

Return the parent element.

Naming Conventions

A type that starts with an `"S"` is *signed*.

A type that starts with an `"U"` is *unsigned*.

A type that has a `"L"` as second character is *little endian*.

A type that has a `"B"` as second character is *big endian*.

8-Bit Types

SLInt8

Signed, little endian 8-Bit integer.

```
class oser.SLInt8(value: int = 0, format: str | None = None)
```

Signed, little endian 8-Bit integer.

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.

Usage:

```
>>> from oser import SLInt8, to_hex

>>> instance = SLInt8(-10)

>>> print(instance)
-10 (SLInt8)

>>> print(instance.introspect())
   0 \xf6  -10 (SLInt8)

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0
\xF6
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
1
>>> print(instance)
-10 (SLInt8)
```

SBInt8

Signed, big endian 8-Bit integer.

```
class oser.SBInt8(value: int = 0, format: str | None = None)
```

Signed, big endian 8-Bit integer.

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.

Usage:

```
>>> from oser import SBInt8, to_hex

>>> instance = SBInt8(-10)

>>> print(instance)
-10 (SBInt8)

>>> print(instance.introspect())
   0 \xf6  -10 (SBInt8)

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0
```

```
\xF6
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
1
>>> print(instance)
-10 (SBInt8)
```

ULInt8

Unsigned, little endian 8-Bit integer.

```
class oser.ULInt8(value: int = 0, format: str | None = None)
```

Unsigned, little endian 8-Bit integer.

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.

Usage:

```
>>> from oser import ULInt8, to_hex

>>> instance = ULInt8(10)

>>> print(instance)
10 (ULInt8)

>>> print(instance.introspect())
   0 \x0a  10 (ULInt8)

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0
\x0A
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
1
>>> print(instance)
10 (ULInt8)
```

UBInt8

Unsigned, big endian 8-Bit integer.

```
class oser.UBInt8(value: int = 0, format: str | None = None)
```

Unsigned, big endian 8-Bit integer.

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.

Usage:

```
>>> from oser import UBInt8, to_hex

>>> instance = UBInt8(10)

>>> print(instance)
10 (UBInt8)

>>> print(instance.introspect())
   0 \x0a  10 (UBInt8)

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0
\x0A
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
1
>>> print(instance)
10 (UBInt8)
```

Padding

Unsigned 8-Bit integer as a fixed pattern. The pattern to use as a placeholder can be set.

If *strict* is used the input data is checked while encoding to match the pattern. If input data and pattern do not match a *ValueError* is raised and decoding is stopped immediately.

```
class oser.Padding(value: bytes | str | int = b'\x00', strict: bool = False, format: str |
None = None)
```

Unsigned 8-Bit integer to fill data with a pattern.

Parameters:

- **value=b"\x00"** – the value to be used as the pattern.
- **strict** – if set to True input data is compared to *value*. In case of a mismatch a *ValueError* is raised.
- **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: bytes | str | 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.

Usage:

```
>>> from oser import Padding, to_hex

>>> instance = Padding(value=b"\x00", strict=True)

>>> print(instance)
\x00 (Padding)

>>> print(instance.introspect())
   0 \x00  \x00 (Padding)

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0
\x00
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
1
>>> print(instance)
\x00 (Padding)

>>> instance.decode(b"\xff")
oser.DecodeException: Data could not be decoded!
Parsing object is:
   0 \xff  \xff (Padding)
     ^^^^ value not allowed: expected \x00 but found \xff
```

16-Bit Types

SLInt16

Signed, little endian 16-Bit integer.

```
class oser.SLInt16(value: int = 0, format: str | None = None)
```

Signed, little endian 16-Bit integer.

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.

Usage:

```
>>> from oser import SLInt16, to_hex

>>> instance = SLInt16(-1000)
>>> print(instance)
-1000 (SLInt16)

>>> print(instance.introspect())
   0 \x18  -1000 (SLInt16)
   1 \xfc

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1
\x18\xFC
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
```

```
2
>>> print(instance)
-1000 (SLInt16)
```

SBInt16

Signed, bit endian 16-Bit integer.

```
class oser.SBInt16(value: int = 0, format: str | None = None)
```

Signed, bit endian 16-Bit integer.

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.

Usage:

```
>>> from oser import SBInt16, to_hex

>>> instance = SBInt16(-1000)
>>> print(instance)
-1000 (SBInt16)

>>> print(instance.introspect())
   0 \xfc  -1000 (SBInt16)
   1 \x18

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1
\xFC\x18
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
2
>>> print(instance)
-1000 (SBInt16)
```

ULInt16

Unsigned, little endian 16-Bit integer.

```
class oser.ULInt16(value: int = 0, format: str | None = None)
```

Unsigned, little endian 16-Bit integer.

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.

Usage:

```
>>> from oser import ULInt16, to_hex

>>> instance = ULInt16(1000)
>>> print(instance)
1000 (ULInt16)

>>> print(instance.introspect())
   0 \xe8  1000 (ULInt16)
   1 \x03

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1
\xE8\x03
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
2
>>> print(instance)
1000 (ULInt16)
```

UBInt16

Unsigned, big endian 16-Bit integer.

```
class oser.UBInt16(value: int = 0, format: str | None = None)
```

Unsigned, big endian 16-Bit integer.

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.

Usage:

```
>>> from oser import UBInt16, to_hex

>>> instance = UBInt16(1000)
>>> print(instance)
1000 (UBInt16)

>>> print(instance.introspect())
   0 \x03  1000 (UBInt16)
   1 \xe8

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1
\x03\xE8
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
2
>>> print(instance)
1000 (UBInt16)
```

32-Bit Types

SLInt32

Signed, little endian 32-Bit integer.

```
class oser.SLInt32(value: int = 0, format: str | None = None)
```

Signed, little endian 32-Bit integer.

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.

Usage:

```
>>> from oser import SLInt32, to_hex

>>> instance = SLInt32(-70000)
>>> print(instance)
-70000 (SLInt32)

>>> print(instance.introspect())
   0 \x90  -70000 (SLInt32)
   1 \xee
   2 \xfe
   3 \xff

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3
\x90\xEE\xFE\xFF
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
4
>>> print(instance)
-70000 (SLInt32)
```

SBInt32

Signed, big endian 32-Bit integer.

```
class oser.SBInt32(value: int = 0, format: str | None = None)
```

Signed, big endian 32-Bit integer.

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.

Usage:

```
>>> from oser import SBInt32, to_hex

>>> instance = SBInt32(-70000)
>>> print(instance)
```

```
-70000 (SBInt32)

>>> print(instance.introspect())
   0 \xff  -70000 (SBInt32)
   1 \xfe
   2 \xee
   3 \x90

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3
\xFF\xFE\xEE\x90
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
4
>>> print(instance)
-70000 (SBInt32)
```

ULInt32

Unsigned, little endian 32-Bit integer.

```
class oser.ULInt32(value: int = 0, format: str | None = None)
```

Unsigned, little endian 32-Bit integer.

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.

Usage:

```
>>> from oser import ULInt32, to_hex

>>> instance = ULInt32(70000)
>>> print(instance)
70000 (ULInt32)

>>> print(instance.introspect())
   0 \x70  70000 (ULInt32)
   1 \x11
   2 \x01
   3 \x00

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3
\x70\x11\x01\x00
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
4
>>> print(instance)
70000 (ULInt32)
```

UBInt32

Unsigned, big endian 32-Bit integer.

```
class oser.UBInt32(value: int = 0, format: str | None = None)
```

Unsigned, big endian 32-Bit integer.

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.

Usage:

```
>>> from oser import UBInt32, to_hex

>>> instance = UBInt32(70000)
>>> print(instance)
70000 (UBInt32)

>>> print(instance.introspect())
   0 \x00  70000 (UBInt32)
   1 \x01
   2 \x11
   3 \x70

>>> binary = instance.encode()
```

```
>>> print(to_hex(binary))
   0|  1|  2|  3
\x00\x01\x11\x70
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
4
>>> print(instance)
70000 (UBInt32)
```

64-Bit Types

SLInt64

Signed, little endian 64-Bit integer.

```
class oser.SLInt64(value: int = 0, format: str | None = None)
```

Signed, little endian 64-Bit integer.

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.

Usage:

```
>>> from oser import SLInt64, to_hex

>>> instance = SLInt64(-7000000)
>>> print(instance)
-7000000 (SLInt64)

>>> print(instance.introspect())
   0 \x40  -7000000 (SLInt64)
   1 \x30
   2 \x95
   3 \xff
   4 \xff
   5 \xff
   6 \xff
   7 \xff

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3|  4|  5|  6|  7
\x40\x30\x95\xFF\xFF\xFF\xFF\xFF
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
8
>>> print(instance)
-7000000 (SLInt64)
```

SBInt64

Signed, big endian 64-Bit integer.

```
class oser.SBInt64(value: int = 0, format: str | None = None)
```

Signed, big endian 64-Bit integer.

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.

Usage:

```
>>> from oser import SBInt64, to_hex

>>> instance = SBInt64(-7000000)
>>> print(instance)
-7000000 (SBInt64)

>>> print(instance.introspect())
   0 \xff  -7000000 (SBInt64)
   1 \xff
   2 \xff
```

```
   3 \xff
   4 \xff
   5 \x95
   6 \x30
   7 \x40

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3|  4|  5|  6|  7
\xFF\xFF\xFF\xFF\xFF\x95\x30\x40
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
8
>>> print(instance)
-7000000 (SBInt64)
```

ULInt64

Unsigned, little endian 64-Bit integer.

```
class oser.ULInt64(value: int = 0, format: str | None = None)
```

Unsigned, little endian 64-Bit integer.

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.

Usage:

```
>>> from oser import ULInt64, to_hex

>>> instance = ULInt64(7000000)
>>> print(instance)
7000000 (ULInt64)

>>> print(instance.introspect())
   0 \xc0  7000000 (ULInt64)
   1 \xcf
   2 \x6a
   3 \x00
   4 \x00
   5 \x00
   6 \x00
   7 \x00

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3|  4|  5|  6|  7
\xC0\xCF\x6A\x00\x00\x00\x00\x00
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
8
>>> print(instance)
7000000 (ULInt64)
```

UBInt64

Unsigned, big endian 64-Bit integer.

```
class oser.UBInt64(value: int = 0, format: str | None = None)
```

Unsigned, big endian 64-Bit integer.

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.

Usage:

```
>>> from oser import UBInt64, to_hex

>>> instance = UBInt64(7000000)
>>> print(instance)
```

```
7000000 (UBInt64)

>>> print(instance.introspect())
   0 \x00  7000000 (UBInt64)
   1 \x00
   2 \x00
   3 \x00
   4 \x00
   5 \x6a
   6 \xcf
   7 \xc0

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3|  4|  5|  6|  7
\x00\x00\x00\x00\x00\x6A\xCF\xC0
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
8
>>> print(instance)
7000000 (UBInt64)
```

Variable Length Integers

Variable length integers transport `7` bits of information per transferred byte. The msb of the transferred bytes indicate the last byte of a variable length integer. All msbs are set except the last one.

Signed values are first converted into an unsigned representation using the zigzag-encoding. This way only one additional bit must be used for the sign.

Zigzag-encoding is done by multiplying a number by `2`. If the number is negative it is xor'ed by `-1` (all bits set) so that all leading sign bits are inverted.

Negative numbers become odd. Positive numbers become even.

original value	zigzag-encoded value
0	0
-1	1
1	2
-2	3
2	4
-3	5
3	6
...	...

Zigzag-decoding is done by dividing the encoded number by `2`. If the number was odd it is xor'ed by `-1` (all bits set) to convert it back into the normal representation.

UBVarInt

Unsigned, big endian variable length integer.

```
class oser.UBVarInt(value: int = 0, format: str | None = None)
```

Unsigned, big endian variable length integer.

Parameters:

- **value=0** – 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.

Returns:
the encoded binary string.
Return type:
bytes

```
get() → None
```

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.

Usage:

```
>>> from oser import UBVarInt, to_hex

>>> instance = UBVarInt(1)
>>> print(instance)
1 (UBVarInt)

>>> print(instance.introspect())
   0 \x01  1 (UBVarInt)

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0
\x01
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
1
>>> print(instance)
1 (UBVarInt)

>>> instance.set(150)
>>> print(instance.introspect())
   0 \x81  150 (UBVarInt)
   1 \x16

>>> instance.set(15000000000)
>>> print(instance.introspect())
   0 \x80  15000000000 (UBVarInt)
   1 \xb7
   2 \xf0
   3 \xc7
   4 \xac
   5 \x00
```

ULVarInt

Unsigned, little endian variable length integer.

```
class oser.ULVarInt(value: int = 0, format: str | None = None)
```

Unsigned, little endian variable length integer.

Parameters:

- **value=0** – 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.

Returns:
the encoded binary string.
Return type:
bytes

```
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.

Usage:

```
>>> from oser import ULVarInt, to_hex

>>> instance = ULVarInt(1)
>>> print(instance)
1 (ULVarInt)

>>> print(instance.introspect())
   0 \x01  1 (ULVarInt)

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0
\x01
>>> bytes_decoded = instance.decode(binary)
```

```
>>> print(bytes_decoded)
1
>>> print(instance)
1 (ULVarInt)

>>> instance.set(150)
>>> print(instance.introspect())
   0 \x96  150 (ULVarInt)
   1 \x01

>>> instance.set(15000000000)
>>> print(instance.introspect())
   0 \x80  15000000000 (ULVarInt)
   1 \xac
   2 \xc7
   3 \xf0
   4 \xb7
   5 \x00
```

SBVarInt

Signed, big endian variable length integer.

```
class oser.SBVarInt(value: int = 0, format: str | None = None)
```

Signed, big endian variable length integer.

Parameters:

- **value=0** – 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.

Returns:
the encoded binary string.
Return type:
bytes

```
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.

Usage:

```
>>> from oser import SBVarInt, to_hex

>>> instance = SBVarInt(-1)
>>> print(instance)
-1 (SBVarInt)

>>> print(instance.introspect())
   0 \x01  -1 (SBVarInt)

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0
\x01
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
1
>>> print(instance)
-1 (SBVarInt)

>>> instance.set(150)
>>> print(instance.introspect())
   0 \x82  150 (SBVarInt)
   1 \x2c

>>> instance.set(-150)
>>> print(instance.introspect())
   0 \x82  -150 (SBVarInt)
   1 \x2b

>>> instance.set(15000000000)
>>> print(instance.introspect())
   0 \x80  15000000000 (SBVarInt)
```

```
   1 \xef
   2 \xe1
   3 \x8e
   4 \xd8
   5 \x00

>>> instance.set(-15000000000)
>>> print(instance.introspect())
   0 \x80  -15000000000 (SBVarInt)
   1 \xef
   2 \xe1
   3 \x8e
   4 \xd7
   5 \x7f
```

SLVarInt

Signed, little endian variable length integer.

```
class oser.SLVarInt(value: int = 0, format: str | None = None)
```

Signed, little endian variable length integer.

Parameters:

- **value=0** – 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.

Returns:
the encoded binary string.
Return type:
bytes

```
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.

Usage:

```
>>> from oser import SLVarInt, to_hex

>>> instance = SLVarInt(-1)
>>> print(instance)
-1 (SLVarInt)

>>> print(instance.introspect())
   0 \x01  -1 (SLVarInt)

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0
\x01
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
1
>>> print(instance)
-1 (SLVarInt)

>>> instance.set(150)
>>> print(instance.introspect())
   0 \xac  150 (SLVarInt)
   1 \x02

>>> instance.set(-150)
>>> print(instance.introspect())
   0 \xab  -150 (SLVarInt)
   1 \x02

>>> instance.set(15000000000)
>>> print(instance.introspect())
   0 \x80  15000000000 (SLVarInt)
   1 \xd8
   2 \x8e
   3 \xe1
```

```
   4 \xef
   5 \x00

>>> instance.set(-15000000000)
>>> print(instance.introspect())
   0 \xff  -15000000000 (SLVarInt)
   1 \xd7
   2 \x8e
   3 \xe1
   4 \xef
   5 \x00
```

Float Types

LFloat

Little endian 32-Bit float.

```
class oser.LFloat(value: float = 0.0)
```

Little endian 32-Bit float.

Parameters:
value – float = 0.0: the initial value.

```
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() → float
```

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: float) → 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.

Usage:

```
>>> from oser import LFloat, to_hex

>>> instance = LFloat(-12345)
>>> print(instance)
-12345 (LFloat)

>>> print(instance.introspect())
   0 \x00  -12345 (LFloat)
   1 \xe4
   2 \x40
   3 \xc6

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3
\x00\xE4\x40\xC6
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
4
>>> print(instance)
-12345.0 (LFloat)
```

BFloat

Big endian 32-Bit float.

```
class oser.BFloat(value: float = 0.0)
```

Big endian 32-Bit float.

Parameters:
value – float = 0.0: the initial value.

```
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() → float
```

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: float) → 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.

Usage:

```
>>> from oser import BFloat, to_hex

>>> instance = BFloat(-12345)
>>> print(instance)
-12345 (BFloat)

>>> print(instance.introspect())
   0 \xc6  -12345 (BFloat)
   1 \x40
   2 \xe4
   3 \x00

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3
\xC6\x40\xE4\x00
```

```
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
4
>>> print(instance)
-12345.0 (BFloat)
```

Double Types

LDouble

Little endian 64-Bit float.

```
class oser.LDouble(value: float = 0.0)
```

Little endian 64-Bit float.

Parameters:
value – float = 0.0: the initial value.

```
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() → float
```

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: float) → 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.

Usage:

```
>>> from oser import LDouble, to_hex

>>> instance = LDouble(-12345)
>>> print(instance)
-12345 (LDouble)

>>> print(instance.introspect())
   0 \x00  -12345 (LDouble)
   1 \x00
   2 \x00
   3 \x00
   4 \x80
   5 \x1c
   6 \xc8
   7 \xc0

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3|  4|  5|  6|  7
\x00\x00\x00\x00\x80\x1C\xC8\xC0
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
8
>>> print(instance)
-12345.0 (LDouble)
```

BDouble

Big endian 64-Bit float.

```
class oser.BDouble(value: float = 0.0)
```

Big endian 64-Bit float.

Parameters:
value – float = 0.0: the initial value.

```
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() → float
```

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: float) → 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.

Usage:

```
>>> from oser import BDouble, to_hex

>>> instance = BDouble(-12345)
>>> print(instance)
-12345 (BDouble)

>>> print(instance.introspect())
   0 \xc0  -12345 (BDouble)
   1 \xc8
   2 \x1c
   3 \x80
   4 \x00
   5 \x00
   6 \x00
   7 \x00

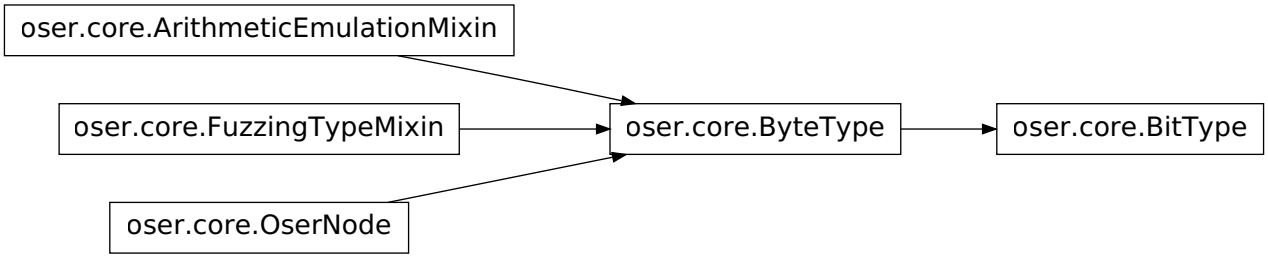
>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3|  4|  5|  6|  7
\xC0\xC8\x1C\x80\x00\x00\x00\x00
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
```

```
8
>>> print(instance)
-12345.0 (BDouble)
```

BitType

`oser.BitType` is an abstract super class which can be used to create bit types.

`oser.BitType` should not be used alone but within a `oser.BitStruct` as a member since `oser.BitType` decodes and encodes bit-strings and not byte-strings.




```
class oser.BitType(value: Any = 0, format: str | None = None)
```

A `BitType` is an abstract class for bit types.

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 bits 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() → Any
```

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 size in bits.

Returns:
the length of the bit type in bits.
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: 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 – the values used for fuzzing.

```
up() → ByteStruct | BitStruct
```

Return the parent element.

Bit Types

Flag

1-Bit value used as a flag in a bit context.

```
class oser.Flag(value: bool | int = False, format: str | None = None)
```

1-Bit value used as a flag in a bit context.

Parameters:

- **value=False** – can be `True` or `False`.
- **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 bits 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 bits 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() → bool
```

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 size in bits.

Returns:
the length of the bit type in bits.
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: bool | 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.

Usage:

```
>>> from oser import Flag

>>> instance = Flag(1)
>>> print(instance)
1 (Flag)

>>> print(instance.introspect())
   0.7  1  1 (Flag)

>>> bit_string = instance.encode()
>>> print(bit_string)
1
>>> bitsDecoded = instance.decode(bit_string)
>>> print(bitsDecoded)
1
>>> print(instance)
1 (Flag)
```

Nibble

4-Bit value in a bit context.

```
class oser.Nibble(value: int = 0, format: str | None = None)
```

4-Bit value in a bit context.

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 bits 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 bits 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() → 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 size in bits.

Returns:
the length of the bit type in bits.
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.

Usage:

```
>>> from oser import Nibble

>>> instance = Nibble(7)
>>> print(instance)
7 (Nibble)

>>> print(instance.introspect())
   0.7  0  7 (Nibble)
   0.6  1
   0.5  1
   0.4  1

>>> bit_string = instance.encode()
>>> print(bit_string)
0111
>>> bitsDecoded = instance.decode(bit_string)
>>> print(bitsDecoded)
4
>>> print(instance)
7 (Nibble)
```

BitField

N-Bit value with a variable length in a bit context.

```
class oser.BitField(value: int = 0, length: int = 1, format: str | None = None)
```

N-Bit value with a variable length in a bit context.

Initialize a `BitField`.

Parameters:

- **value=0** – the initial value
- **length=1** – the length in bits
- **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 bits 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 bits 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 size in bits.

Returns:
the length of the bit type in bits.
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.

Usage:

```
>>> from oser import BitField

>>> instance = BitField(value=0xabc, length=12)
>>> print(instance)
2748 (BitField(12))

>>> print(instance.introspect())
   0.7  1  2748 (BitField(12))
   0.6  0
   0.5  1
   0.4  0
   0.3  1
   0.2  0
   0.1  1
   0.0  1
   1.7  1
   1.6  1
   1.5  0
   1.4  0

>>> bit_string = instance.encode()
>>> print(bit_string)
101010111100
>>> bitsDecoded = instance.decode(bit_string)
>>> print(bitsDecoded)
```

```
12
>>> print(instance)
2748 (BitField(12))
```

PaddingFlag

1-Bit padding value used as a flag in a bit context. Value is fixed and may be strict. If strict is set the input data and the value are compared with each other. If the values differe a *ValueError* is raised.

```
class oser.PaddingFlag(value: bool = False, strict: bool = False, format: str | None = None)
```

PaddingFlag is a one bit padding.

Parameters:

- **value=False** – can be `True` or `False`.
- **strict=False** – if set to `True` an exception is raised when decoded data does not match `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 bits 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 bits 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() → bool
```

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 size in bits.

Returns:
the length of the bit type in bits.
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: bool) → 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.

Usage:

```
>>> from oser import PaddingFlag

>>> instance = PaddingFlag(value=1, strict=True)
>>> print(instance)
1 (PaddingFlag)

>>> print(instance.introspect())
   0.7  1  1 (PaddingFlag)

>>> bit_string = instance.encode()
>>> print(bit_string)
1
>>> bitsDecoded = instance.decode(bit_string)
>>> print(bitsDecoded)
1
>>> print(instance)
1 (PaddingFlag)

>>> instance.decode(b"0")
Exception: Padding error: expected 1 got 0!
```

Null

A null object. It does not lead to any data but can be used as a placeholder or in case of an unused else branch, etc.

```
class oser.Null
```

A null-object that does not contain any data.

Parameters:

- **value** – the initial value.
- **format=None** (*string*) – 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 (always 0).

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. This is always an empty 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 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 (*object*) – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
set(value: Any) → None
```

Set the value. Always raises an Exception.

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.

Usage:

```
>>> from oser import Null, to_hex

>>> instance = Null()
>>> print(instance)
```

```
Null (Null)

>>> print(instance.introspect())
   -    -  Null

>>> binary = instance.encode()
>>> print(to_hex(binary))
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
0
>>> print(instance)
Null (Null)
```

Nothing

An invisible object. It can be used to hide members in special conditions.

```
class oser.Nothing
```

`Nothing` does not appear in the encoded data and not in the string or introspection.

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. This is always an empty 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() → None
```

Always returns `None`.

Returns:
None

```
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: Any) → None
```

Set the value. Always raises an Exception.

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.

Usage:

```
>>> from oser import Nothing, to_hex

>>> instance = Nothing()
>>> print(instance)

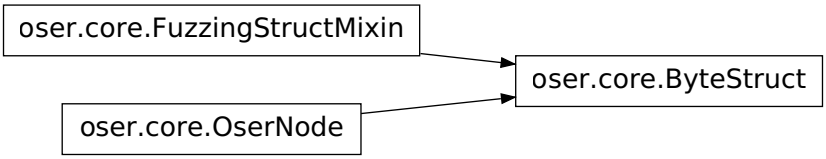
>>> print(instance.introspect())

>>> binary = instance.encode()
>>> print(to_hex(binary))

>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
0
>>> print(instance)
```

Complex Types

ByteStruct




```
class oser.ByteStruct
```

```
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

Usage:

```
>>> from oser import ByteStruct, UBInt8, UBInt16, UBInt32, to_hex

>>> class Data(ByteStruct):
...     def __init__(self):
...         super(Data, self).__init__()
...
...         self.a = UBInt8(1)
...         self.b = UBInt16(1000)
...         self.c = UBInt32(1000000)
```

```
>>> instance = Data()
>>> print(instance)
Data():
    a: 1 (UBInt8)
    b: 1000 (UBInt16)
    c: 1000000 (UBInt32)

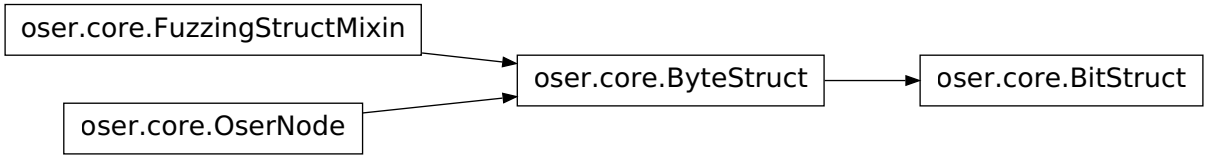
>>> print(instance.introspect())
   -    -  Data():
   0 \x01      a: 1 (UBInt8)
   1 \x03      b: 1000 (UBInt16)
   2 \xe8
   3 \x00      c: 1000000 (UBInt32)
   4 \x0f
   5 \x42
   6 \x40

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3|  4|  5|  6
\x01\x03\xE8\x00\x0F\x42\x40
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
7
>>> print(instance)
Data():
    a: 1 (UBInt8)
    b: 1000 (UBInt16)
    c: 1000000 (UBInt32)
```

BitStruct

`oser.BitStruct` is like a `oser.ByteStruct` for bit-aligned types.

Note: The result of `oser.OserNode.encode()` is byte-aligned. If the amount of bits is not a multiple of 8 the last bits are filled with don't care values.




```
class oser.BitStruct
```

```
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.

```
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 bits.

Returns:
the length of the bit type in bits.
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

Usage:

```
>>> from oser import BitStruct, Flag, Nibble, BitField, to_hex

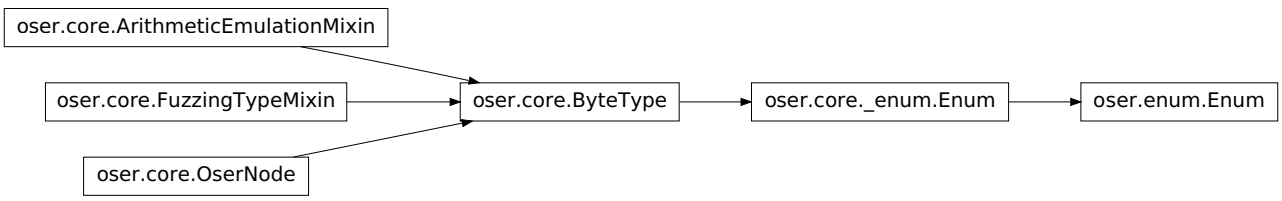
>>> class Data(BitStruct):
...     def __init__(self):
...         super(Data, self).__init__()
...
...         self.a = Flag(1)
...         self.b = Nibble(5)
...         self.c = BitField(value=57, length=6)
...
>>> instance = Data()
>>> print(instance)
Data():
    a: 1 (Flag)
    b: 5 (Nibble)
    c: 57 (BitField(6))
```

```
>>> print(instance.introspect())
   -    -  Data():
   0.7  1      a: 1 (Flag)
   0.6  0      b: 5 (Nibble)
   0.5  1
   0.4  0
   0.3  1
   0.2  1      c: 57 (BitField(6))
   0.1  1
   0.0  1
   1.7  0
   1.6  0
   1.5  1
   1.4  x
   1.3  x
   1.2  x
   1.1  x
   1.0  x

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1
\xAF\x20
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
2
>>> print(instance)
Data():
    a: 1 (Flag)
    b: 5 (Nibble)
    c: 57 (BitField(6))
```

Enum

An `oser.Enum` is most commonly used to give values human readable names. It can be used in bit-aligned context and byte-aligned context.




```
class oser.Enum(prototype: ByteStruct | BitStruct | ByteType | BitType, values:
Dict[ByteStruct | BitStruct | ByteType | BitType, Any], value: Any | None = None, strict:
bool = False)
```

An enumeration serializer.

Parameters:

- **prototype** – the class of the underlying data type.
- **values** – a dictionary of key-value-pairs. The keys are translated into binary data represented by the values.
- **value=None** (*object*) – the initial value.
- **strict=False** (*bool*) – If set to *True* only keys and values appearing in *values* are accepted. Else all values are accepted.

```
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() → Any
```

Return the value as string representation if possible of the raw value else.

Returns:
the value as string or raw
Return type:
object

```
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`.

```
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.

Usage in byte context:

```
>>> from oser import ByteStruct, Enum, UBInt16, to_hex

>>> class Data(ByteStruct):
...         def __init__(self):
...             super(Data, self).__init__()
...             self.enum = Enum(prototype=UBInt16,
...                              values={
...                                  "A": 1,
...                                  "B": 2,
...                                  "C": 3,
...                                  "D": 4,
...                              }, value="C")
...
>>> instance = Data()
>>> print(instance)
Data():
    enum: 'C' (UBInt16)

>>> print(instance.introspect())
   -    -  Data():
   0 \x00      enum: 3 (UBInt16)
   1 \x03

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1
\x00\x03
>>> instance.enum.set("B")
>>> print(instance)
Data():
    enum: 'B' (UBInt16)

>>> print(instance.introspect())
   -    -  Data():
   0 \x00      enum: 2 (UBInt16)
   1 \x02

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1
\x00\x02
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
2
>>> print(instance)
Data():
    enum: 'B' (UBInt16)

>>> instance.decode(b"\x01\x4D")
2
```

```
>>> print(instance)
Data():
    enum: '333' (UBInt16)

>>> print(instance.introspect())
   -    -  Data():
   0 \x01      enum: 333 (UBInt16)
   1 \x4d
```

Usage in bit context:

```
>>> from oser import BitStruct, Enum, Flag, to_hex

>>> class Data(BitStruct):
...         def __init__(self):
...             super(Data, self).__init__()
...             self.enum = Enum(prototype=Flag,
...                              values={
...                                  "false" : 0,
...                                  "true" : 1,
...                              }, value="true")
...
>>> instance = Data()
>>> print(instance)
Data():
    enum: 'true' (Flag)

>>> print(instance.introspect())
   -    -  Data():
   0.7  1      enum: 1 (Flag)
   0.6  x
   0.5  x
   0.4  x
   0.3  x
   0.2  x
   0.1  x
   0.0  x

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0
\x80

>>> instance.enum.set("false")
>>> print(instance)
Data():
    enum: 'false' (Flag)

>>> print(instance.introspect())
   -    -  Data():
   0.7  0      enum: 0 (Flag)
   0.6  x
   0.5  x
   0.4  x
   0.3  x
   0.2  x
   0.1  x
```

```
   0.0  x

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0
\x00
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
1
>>> print(instance)
Data():
    enum: 'false' (Flag)


>>> instance.decode(b"\x80")
1
>>> print(instance)
Data():
    enum: 'true' (Flag)

>>> print(instance.introspect())
   -    -  Data():
   0.7  1      enum: 1 (Flag)
   0.6  x
   0.5  x
   0.4  x
   0.3  x
   0.2  x
   0.1  x
   0.0  x
```

Context Based Types

The Context

Every OSER instance is aware of the context it is located in. Context based types can use the context to realize context dependent instances. For example it is possible to create a variable length payload for a network packet or create one instance that can represent multiple different network packets, etc. .

The context can be used using a lambda-expression or a method.

String

A `oser.String` can be used to encode and decode strings. The length can be either fixed (`int`), variable (`callable`) or `None` for null-terminated strings.

```
class oser.String(length: Callable | int | None = None, value: bytes | str = b'', padding:
bytes | str = b'\x00')
```

`String` builds a string serializer with a fixed or variable 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`.
- **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.

Fixed length usage:

```
>>> from oser import String, to_hex

>>> instance = String(length=10, value=b"abcdefghi")
>>> print(instance)
'abcdefghi'

>>> print(instance.introspect())
   -    -  String():
   0 \x61      'a'
   1 \x62      'b'
   2 \x63      'c'
   3 \x64      'd'
   4 \x65      'e'
   5 \x66      'f'
   6 \x67      'g'
   7 \x68      'h'
   8 \x69      'i'
   9 \x00      '\x00'

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x61\x62\x63\x64\x65\x66\x67\x68\x69\x00
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
10
>>> print(instance)
'abcdefghi\x00'
```

Variable length usage in a `oser.ByteStruct`:

```
>>> from oser import ByteStruct, UBInt16, String, to_hex

>>> class Data(ByteStruct):
...     def __init__(self):
...         super(Data, self).__init__()
...
...         self.length = UBInt16(12)
...         self.string = String(length=lambda self: self.length.get(), value="abcdefghijkl")
...
>>> instance = Data()
>>> print(instance)
Data():
    length: 12 (UBInt16)
    string: 'abcdefghijkl'

>>> print(instance.introspect())
   -    -  Data():
   0 \x00      length: 12 (UBInt16)
   1 \x0c
   -    -      string: String():
   2 \x61          'a'
   3 \x62          'b'
   4 \x63          'c'
```

```
   5 \x64          'd'
   6 \x65          'e'
   7 \x66          'f'
   8 \x67          'g'
   9 \x68          'h'
  10 \x69          'i'
  11 \x6a          'j'
  12 \x6b          'k'
  13 \x6c          'l'

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13
\x00\x0C\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6A\x6B\x6C
>>> bytes_decoded =
instance.decode(b"\x00\x1A\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6A\x6B\x6C\x6D\x6E\x6F\x70\x71\x72\x73\
>>> print(bytes_decoded)
28
>>> print(instance)
Data():
    length: 26 (UBInt16)
    string: 'abcdefghijklmnopqrstuvwxyz'
```

Null-terminated `oser.String`:

```
>>> from oser import String, to_hex

>>> instance = String(length=None, value=b"abc")
>>> print(instance)
'abc'

>>> print(instance.introspect())
   -    -  String():
   0 \x61      'a'
   1 \x62      'b'
   2 \x63      'c'
   3 \x00      '\x00'

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3
\x61\x62\x63\x00
>>> bytes_decoded = instance.decode(b"abcdefg\x00this text will not be parsed!")
>>> print(bytes_decoded)
8
>>> print(instance)
'abcdefg'

>>> print(instance.size())
8
>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3|  4|  5|  6|  7
\x61\x62\x63\x64\x65\x66\x67\x00
```

Data

A `oser.Data` can be used to encode and decode data (strings). The length can be either fixed or variable.

```
class oser.Data(length: Callable | int, value: bytes | str = b'', padding: bytes | str =
b'\x00')
```

`Data` builds a data serializer with fixed or variable length. Extends `oser.String` but length must not be `None`.

Parameters:

- **length** – states the data length. Must be a callable (e.g. lambda) or an integer.
- **value=""** – the initial data value.
- **padding=b"\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** (*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() → 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 (*object*) – 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 (*iterable*) – 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.

Array

An `oser.Array` can be used to repeat parts of an instance. The length can be either fixed or variable.

```
class oser.Array(length: Callable | int, prototype: ByteStruct | BitStruct | ByteType |
BitType, args: Tuple | None = None, kwargs: Dict | None = None, values: List[ByteStruct |
BitStruct | ByteType | BitType] | None = None)
```

`Array` builds an array serializer with a fixed or variable length.

Parameters:

- **length** – states the array length. Can be a callable (e.g. lambda) or a scalar.
- **prototype** – a class that is the prototype for *values*
- **args=None** – a tuple of positional arguments that are passed to *prototype* when it is instantiated when array size is extended. Note: don't forget a leading comma when passing one value `(1,)` (`(1)` is an integer!).
- **kwargs=None** – a dictionary of named arguments that are passed to *prototype* when it is instantiated when array size is extended.
- **values=None** – the initial array values. Can be left empty. If empty the array is automatically resized with instances of `prototype(*args, **kwargs)`.

```
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

```
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() → Array
```

Return the value.

Returns:
the value

```
get_byte_size() → int
```

Return the current size in bytes.

```
get_size() → int
```

Return the current size in bytes or bits according to the context.

```
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() → List[Dict[Any, Any]]
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Fixed length usage:

```
>>> from oser import Array, UBInt16, to_hex

>>> instance = Array(length=3, prototype=UBInt16, values=[UBInt16(ii) for ii in range(3)])
>>> print(instance)
Array():
[
    @0: 0 (UBInt16)
    @1: 1 (UBInt16)
    @2: 2 (UBInt16)
]

>>> print(instance.introspect())
   -    -  Array():
   -    -  [
   0 \x00      @0: 0 (UBInt16)
   1 \x00
   2 \x00      @1: 1 (UBInt16)
   3 \x01
   4 \x00      @2: 2 (UBInt16)
   5 \x02
   -    -  ]
```

```
>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3|  4|  5
\x00\x00\x00\x01\x00\x02
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
6
>>> print(instance)
Array():
[
    @0: 0 (UBInt16)
    @1: 1 (UBInt16)
    @2: 2 (UBInt16)
]

>>> print(instance[10])
IndexError: list index out of range
```

Variable length usage:

```
>>> from oser import ByteStruct, Array, UBInt16, to_hex

>>> class Data(ByteStruct):
...     def __init__(self):
...         super(Data, self).__init__()
...
...         self.length = UBInt16(4)
...         self.data = Array(length=lambda self: self.length.get(),
...                           prototype=UBInt16,
...                           args=(1337,), # when range is extended
...                                         # instantiate prototype with this arguments
...                           values=[UBInt16(ii) for ii in range(2)])
...
>>> instance = Data()
>>> print(instance)
Data():
    length: 4 (UBInt16)
    data: Array():
    [
        @0: 0 (UBInt16)
        @1: 1 (UBInt16)
        @2: 1337 (UBInt16)
        @3: 1337 (UBInt16)
    ]

>>> print(instance.introspect())
   -    -  Data():
   0 \x00      length: 4 (UBInt16)
   1 \x04
   -    -      data: Array():
   -    -      [
   2 \x00          @0: 0 (UBInt16)
   3 \x00
   4 \x00          @1: 1 (UBInt16)
   5 \x01
   6 \x05          @2: 1337 (UBInt16)
   7 \x39
```

```
   8 \x05          @3: 1337 (UBInt16)
   9 \x39
   -    -      ]

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x00\x04\x00\x00\x00\x01\x05\x39\x05\x39
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
10
>>> print(instance)
Data():
    length: 4 (UBInt16)
    data: Array():
    [
        @0: 0 (UBInt16)
        @1: 1 (UBInt16)
        @2: 1337 (UBInt16)
        @3: 1337 (UBInt16)
    ]

>>> print(instance.introspect())
   -    -  Data():
   0 \x00      length: 4 (UBInt16)
   1 \x04
   -    -      data: Array():
   -    -      [
   2 \x00          @0: 0 (UBInt16)
   3 \x00
   4 \x00          @1: 1 (UBInt16)
   5 \x01
   6 \x05          @2: 1337 (UBInt16)
   7 \x39
   8 \x05          @3: 1337 (UBInt16)
   9 \x39
   -    -      ]

>>> print(instance.data[3])
1337 (UBInt16)

>>>
instance.decode(b"\x00\x14\x00\x00\x00\x01\x00\x02\x00\x03\x00\x04\x00\x05\x00\x06\x00\x07\x00\x08\x00\
42
>>> print(instance)
Data():
    length: 20 (UBInt16)
    data: Array():
    [
        @0: 0 (UBInt16)
        @1: 1 (UBInt16)
        @2: 2 (UBInt16)
        @3: 3 (UBInt16)
        @4: 4 (UBInt16)
        @5: 5 (UBInt16)
        @6: 6 (UBInt16)
        @7: 7 (UBInt16)
        @8: 8 (UBInt16)
        @9: 9 (UBInt16)
        @10: 10 (UBInt16)
```

```
        @11: 11 (UBInt16)
        @12: 12 (UBInt16)
        @13: 13 (UBInt16)
        @14: 14 (UBInt16)
        @15: 15 (UBInt16)
        @16: 16 (UBInt16)
        @17: 17 (UBInt16)
        @18: 18 (UBInt16)
        @19: 19 (UBInt16)
    ]

>>> print(instance.introspect())
   -    -  Data():
   0 \x00      length: 20 (UBInt16)
   1 \x14
   -    -      data: Array():
   -    -      [
   2 \x00          @0: 0 (UBInt16)
   3 \x00
   4 \x00          @1: 1 (UBInt16)
   5 \x01
   6 \x00          @2: 2 (UBInt16)
   7 \x02
   8 \x00          @3: 3 (UBInt16)
   9 \x03
  10 \x00          @4: 4 (UBInt16)
  11 \x04
  12 \x00          @5: 5 (UBInt16)
  13 \x05
  14 \x00          @6: 6 (UBInt16)
  15 \x06
  16 \x00          @7: 7 (UBInt16)
  17 \x07
  18 \x00          @8: 8 (UBInt16)
  19 \x08
  20 \x00          @9: 9 (UBInt16)
  21 \x09
  22 \x00          @10: 10 (UBInt16)
  23 \x0a
  24 \x00          @11: 11 (UBInt16)
  25 \x0b
  26 \x00          @12: 12 (UBInt16)
  27 \x0c
  28 \x00          @13: 13 (UBInt16)
  29 \x0d
  30 \x00          @14: 14 (UBInt16)
  31 \x0e
  32 \x00          @15: 15 (UBInt16)
  33 \x0f
  34 \x00          @16: 16 (UBInt16)
  35 \x10
  36 \x00          @17: 17 (UBInt16)
  37 \x11
  38 \x00          @18: 18 (UBInt16)
  39 \x12
  40 \x00          @19: 19 (UBInt16)
  41 \x13
   -    -      ]
```

```
>>> print(instance.data[15])
15 (UBInt16)
```

Setting and getting items:

```
>>> from oser import ByteStruct, Array, UBInt16

>>> class Data(ByteStruct):
...     def __init__(self):
...         super(Data, self).__init__()
...
...         self.length = UBInt16(4)
...         self.data = Array(length=lambda self: self.length.get(),
...                           prototype=UBInt16,
...                           values=[UBInt16(ii) for ii in range(2)])
...
>>> instance = Data()
>>> print(instance)
Data():
    length: 4 (UBInt16)
    data: Array():
    [
        @0: 0 (UBInt16)
        @1: 1 (UBInt16)
        @2: 0 (UBInt16)
        @3: 0 (UBInt16)
    ]


>>> # set a single item
... instance.data[1] = 23
>>> print(instance)
Data():
    length: 4 (UBInt16)
    data: Array():
    [
        @0: 0 (UBInt16)
        @1: 23 (UBInt16)
        @2: 0 (UBInt16)
        @3: 0 (UBInt16)
    ]


>>> # set multiple items (with a slice)
... instance.data[:] = [1, 2, 3, 4]
>>> print(instance)
Data():
    length: 4 (UBInt16)
    data: Array():
    [
        @0: 1 (UBInt16)
        @1: 2 (UBInt16)
        @2: 3 (UBInt16)
        @3: 4 (UBInt16)
    ]
```

```
>>> # set every second item
... instance.data[::2] = [23, 24]
>>> print(instance)
Data():
    length: 4 (UBInt16)
    data: Array():
    [
        @0: 23 (UBInt16)
        @1: 2 (UBInt16)
        @2: 24 (UBInt16)
        @3: 4 (UBInt16)
    ]


>>> # get a single item
... print(instance.data[3])
4 (UBInt16)


>>> # get a slice
... print(instance.data[0:2])
[<oser.typedef.UBInt16 object at 0x7f8d8a86a748>, <oser.typedef.UBInt16 object at 0x7f8d8a86d470>]
```

If

`oser.If` is used to build conditional parsers when the condition is *True* or *False*. Compared to `IfElse` `If` is invisble if the condition is *False*.

```
class oser.If(condition: Callable | bool, if_true: ByteStruct | BitStruct | ByteType |
BitType)
```

`If` builds a conditional serializer for boolean conditions that is invisible if the condition is `False`.

It is an alias for `IfElse(condition=condition, if_true=if_true, if_false=Nothing())`

Parameters:

- **condition** – the condition which selects the desired values. Can be a callable (e.g. lambda) or a scalar.
- **if_true** – is used if *condition* is *True*.

```
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

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data (*dict*) – 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 (*bool*) – 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() → ByteStruct | BitStruct | ByteType | BitType
```

Return the current value.

```
get_byte_size() → int
```

Return the current size in bytes.

```
get_current() → ByteStruct | BitStruct | ByteType | BitType
```

Return the currently selected object.

```
get_false_value() → ByteStruct | BitStruct | ByteType | BitType
```

Return the value for a `False` `condition`.

```
get_size() → int
```

Return the current size in bytes or bits according to the context.

```
get_true_value() → ByteStruct | BitStruct | ByteType | BitType
```

Return the value for a `True` `condition`.

```
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`.

```
root() → ByteStruct | BitStruct
```

return root element

```
set(value: ByteStruct | BitStruct | ByteType | BitType) → None
```

Set the value.

Parameters:
value – the new value

```
set_false_value(value: ByteStruct | BitStruct | ByteType | BitType) → None
```

Set the value for a `False` `condition`.

Parameters:

value – the new value for a `False` `condition`.

```
set_true_value(value: ByteStruct | BitStruct | ByteType | BitType) → None
```

Set the value for a `True` `condition`.

Parameters:
value – the new value for a `True` `condition`.

```
to_dict() → Dict[Any, Any] | List[Dict[Any, Any]]
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Usage:

```
>>> from oser import ByteStruct, If, UBInt8, UBInt64, to_hex

>>> class Data(ByteStruct):
...     def __init__(self):
...         super(Data, self).__init__()
...
...         self.condition = UBInt8(1)
...         self.data = If(condition=lambda self: self.condition.get() > 0,
...                        if_true=UBInt8(1))
...
>>> instance = Data()
>>> print(instance)
Data():
    condition: 1 (UBInt8)
    data: 1 (UBInt8)

>>> print(instance.introspect())
   -    -  Data():
   0 \x01      condition: 1 (UBInt8)
   1 \x01      data: 1 (UBInt8)

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1
\x01\x01
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
2
>>> print(instance)
Data():
    condition: 1 (UBInt8)
    data: 1 (UBInt8)
```

```
>>> print(instance.introspect())
   -    -  Data():
   0 \x01      condition: 1 (UBInt8)
   1 \x01      data: 1 (UBInt8)

>>> instance.condition.set(0)
>>> print(instance)
Data():
    condition: 0 (UBInt8)

>>> print(instance.introspect())
   -    -  Data():
   0 \x00      condition: 0 (UBInt8)

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0
\x00
```

IfElse

`oser.IfElse` is used to build conditional parsers when the condition is *True* or *False*. If there are more possible values `oser.Switch` can be used.

```
class oser.IfElse(condition: Callable | bool, if_true: ByteStruct | BitStruct | ByteType |
BitType, if_false: ByteStruct | BitStruct | ByteType | BitType)
```

`IfElse` builds a conditional serializer for boolean conditions.

Parameters:

- **condition** – the condition which selects the desired values. Can be a callable (e.g. lambda) or a scalar.
- **if_true** – is used if *condition* is *True*.
- **if_false** – is used if *condition* is *False*.

```
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

```
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() → ByteStruct | BitStruct | ByteType | BitType
```

Return the current value.

```
get_byte_size() → int
```

Return the current size in bytes.

```
get_current() → ByteStruct | BitStruct | ByteType | BitType
```

Return the currently selected object.

```
get_false_value() → ByteStruct | BitStruct | ByteType | BitType
```

Return the value for a `False` `condition`.

```
get_size() → int
```

Return the current size in bytes or bits according to the context.

```
get_true_value() → ByteStruct | BitStruct | ByteType | BitType
```

Return the value for a `True` `condition`.

```
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: ByteStruct | BitStruct | ByteType | BitType) → None
```

Set the value.

Parameters:
value – the new value

```
set_false_value(value: ByteStruct | BitStruct | ByteType | BitType) → None
```

Set the value for a `False` `condition`.

Parameters:
value – the new value for a `False` `condition`.

```
set_true_value(value: ByteStruct | BitStruct | ByteType | BitType) → None
```

Set the value for a `True` `condition`.

Parameters:
value – the new value for a `True` `condition`.

```
to_dict() → Dict[Any, Any] | List[Dict[Any, Any]]
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Usage:

```
>>> from oser import ByteStruct, IfElse, UBInt8, UBInt64, to_hex

>>> class Data(ByteStruct):
...     def __init__(self):
...         super(Data, self).__init__()
...
...         self.condition = UBInt8(1)
...         self.data = IfElse(condition=lambda self: self.condition.get() > 0,
...                            if_true=UBInt8(1),
...                            if_false=UBInt64(0x0123456789abcdef)
...                            )
...
>>> instance = Data()
>>> print(instance)
Data():
    condition: 1 (UBInt8)
    data: 1 (UBInt8)

>>> print(instance.introspect())
   -    -  Data():
   0 \x01      condition: 1 (UBInt8)
   1 \x01      data: 1 (UBInt8)

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1
\x01\x01
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
2
>>> print(instance)
Data():
    condition: 1 (UBInt8)
    data: 1 (UBInt8)

>>> print(instance.introspect())
   -    -  Data():
   0 \x01      condition: 1 (UBInt8)
   1 \x01      data: 1 (UBInt8)
```

```
>>> instance.condition.set(0)
>>> print(instance)
Data():
    condition: 0 (UBInt8)
    data: 81985529216486895 (UBInt64)

>>> print(instance.introspect())
   -    -  Data():
   0 \x00      condition: 0 (UBInt8)
   1 \x01      data: 81985529216486895 (UBInt64)
   2 \x23
   3 \x45
   4 \x67
   5 \x89
   6 \xab
   7 \xcd
   8 \xef

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3|  4|  5|  6|  7|  8
\x00\x01\x23\x45\x67\x89\xAB\xCD\xEF
```

Switch

`oser.Switch` is used to build conditional parsers where condition is not only *True* or *False*. If the condition can only be *True* or *False* `oser.IfElse` can be used, too.

```
class oser.Switch(condition: Callable | int, values: Dict[Any, ByteStruct | BitStruct |
ByteType | BitType], default: ByteStruct | BitStruct | ByteType | BitType | None = None)
```

`Switch` builds a conditional serializer.

Parameters:

- **condition** – the key which selects the desired values. Can be a callable (e.g. lambda) or a scalar.
- **values** – a dictionary that holds key-value pairs. The key is selected by the value (or return value) of *condition* and the value is used as the actual serializer.
- **default=None** – if not *None* this value is used as a serializer if *condition* is not found in *values*

```
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

```
delete_value(key: Any) → None
```

Delete the value identified by `key`.

Parameters:
key – the key for the value to be deleted.

```
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

```
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() → ByteStruct | BitStruct | ByteType | BitType
```

Return the current value.

```
get_byte_size() → int
```

Return the current size in bytes.

```
get_current() → ByteStruct | BitStruct | ByteType | BitType
```

Return the currently selected object.

```
get_items() → Iterable[ByteStruct | BitStruct | ByteType | BitType]
```

Return the items of the values (like in a `dict`).

```
get_keys() → Iterable[Any]
```

Return the keys of the values (like in a `dict`).

```
get_size() → int
```

Return the current size in bytes or bits according to the context.

```
get_value(key: Any) → ByteStruct | BitStruct | ByteType | BitType
```

Get the value for `key`.

Parameters:
key – the key for the condition.
Returns:
the value for the `condition` `key`.

```
get_values() → Iterable[ByteStruct | BitStruct | ByteType | BitType]
```

Return the values of the `values` (like in a `dict`).

```
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: ByteStruct | BitStruct | ByteType | BitType) → None
```

Set the current value.

Parameters:
value – the new value.

```
set_value(key: Any, value: ByteStruct | BitStruct | ByteType | BitType) → None
```

Get the value for `key`.

Parameters:

- **key** – the key for the condition.
- **value** – the value for the `key`.

```
to_dict() → Dict
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

```
update_values(values: Dict[Any, ByteStruct | BitStruct | ByteType | BitType]) → None
```

Update the values. Overwrites or adds new keys and values.

Parameters:
values – the dict with the new values.

Usage:

```
>>> from oser import ByteStruct, Switch, UBInt8, UBInt16, UBInt32, Null, to_hex
```

```
>>> class Data(ByteStruct):
...     def __init__(self):
...         super(Data, self).__init__()
...
...         self.type = UBInt8(1)
...         self.data = Switch(lambda self: self.type.get(),
...                            values={
...                                1: UBInt8(1),
...                                2: UBInt16(2),
...                                3: UBInt32(3)
...                            },
...                            default=Null())
...
>>> instance = Data()
>>> print(instance)
Data():
    type: 1 (UBInt8)
    data: 1 (UBInt8)

>>> print(instance.introspect())
   -    -  Data():
   0 \x01      type: 1 (UBInt8)
   1 \x01      data: 1 (UBInt8)

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1
\x01\x01
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
2
>>> print(instance)
Data():
    type: 1 (UBInt8)
    data: 1 (UBInt8)

>>> print(instance.introspect())
   -    -  Data():
   0 \x01      type: 1 (UBInt8)
   1 \x01      data: 1 (UBInt8)

>>> # 3
... instance.type.set(3)
>>> print(instance)
Data():
    type: 3 (UBInt8)
    data: 3 (UBInt32)

>>> print(instance.introspect())
   -    -  Data():
   0 \x03      type: 3 (UBInt8)
   1 \x00      data: 3 (UBInt32)
   2 \x00
   3 \x00
   4 \x03

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3|  4
\x03\x00\x00\x00\x03
```

```
>>> # non existent value -> Null()
... instance.type.set(100)
>>> print(instance)
Data():
    type: 100 (UBInt8)
    data: Null (Null)

>>> print(instance.introspect())
   -    -  Data():
   0 \x64      type: 100 (UBInt8)
   -    -      data: Null

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0
\x64
```

Value

`oser.Value` evaluates a value that only appears in string or introspection representation but not in the encoded data.

```
class oser.Value(value: Callable | Any, format: str | None = None)
```

`Value` evaluates a value for introspection. It does not appear in the encoded data.

Parameters:

- **value** (*callable*) – a method or a lambda to generate the value.
- **format=None** (*string*) – 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** (*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() → Any
```

Return the value.

Returns:
the value.

```
get_byte_size() → int
```

The size is always 0 because `Value` does not appear in the encoded data.

```
get_size() → int
```

The size is always 0 because `Value` does not appear in the encoded data.

```
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`.

```
root() → ByteStruct | BitStruct
```

return root element

```
set(value: Callable | 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.

Usage:

```
>>> from oser import ByteStruct, ULInt8, ULInt16, to_hex, Value

>>> class Foo(ByteStruct):
...     def __init__(self):
...         super(Foo, self).__init__()
...         self.a = ULInt8(2)
...         self.b = ULInt16(7)
...         self.value = Value(value=lambda self: self.a.get() * self.b.get())
...
>>> foo = Foo()
>>> print(foo)
Foo():
    a: 2 (ULInt8)
    b: 7 (ULInt16)
    value: 14 (Value)

>>> print(foo.introspect())
   -    -  Foo():
   0 \x02      a: 2 (ULInt8)
   1 \x07      b: 7 (ULInt16)
   2 \x00
   -    -      value: 14
>>> binary = foo.encode()
>>> print(to_hex(binary))
   0|  1|  2
\x02\x07\x00
```

Checksum Types

CRC — Cyclic Redundancy Check

A *CRC* calculates a crc over the data in its hierarchical level, e.g. the `oser.ByteStruct` it is located in.

The user can control automatic calculation and checks by setting *crc._automatic_calculation* to `False` at runtime.

CRCL8

8-bit crc little endian.

```
class oser.CRCL8(strict: bool = False, polynomial: int = 213, reflect_input: bool = False,
reflect_output: bool = False, initial_value: int = 0, xor_output: int = 0,
automatic_calculation: bool = True, format: str = 'hex')
```

8-bit crc little endian serializer.

Parameters:

- **strict** – bool = False: If set to `True` crc is checked when decoding data. An exception is raised if crcs do not match.
- **polynomial** – int = 0xd5: the crc polynomial
- **reflect_input** – bool = False: reflect input data
- **reflect_output** – bool = False: reflect output data
- **initial_value** – int = 0x00: start with *initial_value*
- **xor_output** – int = 0x00: xor output data with *xor_output*
- **automatic_calculation** – bool = True: if set to `True` crc is automatically calculated when encoding data
- **format** – str = “hex” (string): Use “hex” to format values to hexadecimal. Use “bin” to format values to binary. Use `None` for decimal format.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string into a byte type 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 byte type as a 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.

```
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 (*object*) – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
set(value: int) → None
```

Set the value.

Parameters:
value – the new value

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

Enable or disable automatic calculation when `encode` is called.

Parameters:
enabled=True (*bool*) – the new enabled state.

```
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.

Usage:

```
>>> from oser import ByteStruct, UBInt16, CRCL8, to_hex

>>> class Data(ByteStruct):
...         def __init__(self):
...             super(Data, self).__init__()
...
...             self.a = ULInt16(1)
...             self.b = ULInt16(2)
...
...             self.crc = CRCL8(strict=True)
...
>>> instance = Data()
>>> print(instance)
Data():
    a: 1 (ULInt16)
    b: 2 (ULInt16)
    crc: 0 (CRCL8)

>>> print(instance.introspect())
   -    -  Data():
   0 \x01      a: 1 (ULInt16)
   1 \x00
   2 \x02      b: 2 (ULInt16)
   3 \x00
   4 \x00      crc: 0 (CRCL8)

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3|  4
\x01\x00\x02\x00\x53
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
```

```
5
>>> print(instance)
Data():
    a: 1 (ULInt16)
    b: 2 (ULInt16)
    crc: 83 (CRCL8)

>>> print(instance.introspect())
   -    -  Data():
   0 \x01      a: 1 (ULInt16)
   1 \x00
   2 \x02      b: 2 (ULInt16)
   3 \x00
   4 \x53      crc: 83 (CRCL8)


>>> bytes_decoded = instance.decode(b"\x01\x00\x02\x00\x53")
>>> print(bytes_decoded)
5
>>> print(instance)
Data():
    a: 1 (ULInt16)
    b: 2 (ULInt16)
    crc: 83 (CRCL8)

>>> print(instance.introspect())
   -    -  Data():
   0 \x01      a: 1 (ULInt16)
   1 \x00
   2 \x02      b: 2 (ULInt16)
   3 \x00
   4 \x53      crc: 83 (CRCL8)


>>> bytes_decoded = instance.decode(b"\x01\x00\x02\x00\x00")  # corrupted crc
oser.ChecksumMismatchException: Data could not be decoded!
Parsing object is:
   -    -  Data():
   0 \x01      a: 1 (ULInt16)
   1 \x00
   2 \x02      b: 2 (ULInt16)
   3 \x00
   4 \x00      crc: 0 (CRCL8)
     ^^^^ Checksum mismatch: expected \x53 but found \x00
```

CRCB8

8-bit crc big endian.

```
class oser.CRCB8(strict: bool = False, polynomial: int = 213, reflect_input: bool = False,
reflect_output: bool = False, initial_value: int = 0, xor_output: int = 0,
automatic_calculation: bool = True, format: str = 'hex')
```

8-bit crc big endian serializer.

Parameters:

- **strict** – bool = False: If set to `True` crc is checked when decoding data. An exception is raised if crcs do not match.
- **polynomial** – int = 0xd5: the crc polynomial
- **reflect_input** – bool = False: reflect input data
- **reflect_output** – bool = False: reflect output data
- **initial_value** – int = 0x00: start with *initial_value*
- **xor_output** – int = 0x00: xor output data with *xor_output*
- **automatic_calculation** – bool = True: if set to `True` crc is automatically calculated when encoding data
- **format** – str = "hex" (string): Use "hex" to format values to hexadecimal. Use "bin" to format values to binary. Use `None` for decimal format.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string into a byte type 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.

```
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() → 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 (*object*) – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
set(value: int) → None
```

Set the value.

Parameters:
value – the new value

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

Enable or disable automatic calculation when `encode` is called.

Parameters:
enabled=True (*bool*) – the new enabled state.

```
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.

Usage:

```
>>> from oser import ByteStruct, UBInt16, CRCB8, to_hex

>>> class Data(ByteStruct):
...         def __init__(self):
...             super(Data, self).__init__()
...
...             self.a = UBInt16(1)
...             self.b = UBInt16(2)
...
...             self.crc = CRCB8(strict=True)
...
>>> instance = Data()
>>> print(instance)
Data():
    a: 1 (UBInt16)
    b: 2 (UBInt16)
    crc: 0 (CRCB8)

>>> print(instance.introspect())
   -    -  Data():
   0 \x00      a: 1 (UBInt16)
   1 \x01
   2 \x00      b: 2 (UBInt16)
   3 \x02
   4 \x00      crc: 0 (CRCB8)

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3|  4
\x00\x01\x00\x02\xFC
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
5
>>> print(instance)
Data():
    a: 1 (UBInt16)
    b: 2 (UBInt16)
    crc: 252 (CRCB8)

>>> print(instance.introspect())
   -    -  Data():
```

```
   0 \x00      a: 1 (UBInt16)
   1 \x01
   2 \x00      b: 2 (UBInt16)
   3 \x02
   4 \xfc      crc: 252 (CRCB8)


>>> bytes_decoded = instance.decode(b"\x00\x01\x00\x02\xFC")
>>> print(bytes_decoded)
5
>>> print(instance)
Data():
    a: 1 (UBInt16)
    b: 2 (UBInt16)
    crc: 252 (CRCB8)

>>> print(instance.introspect())
   -    -  Data():
   0 \x00      a: 1 (UBInt16)
   1 \x01
   2 \x00      b: 2 (UBInt16)
   3 \x02
   4 \xfc      crc: 252 (CRCB8)


>>> bytes_decoded = instance.decode(b"\x00\x01\x00\x02\x00")  # corrupted crc
oser.ChecksumMismatchException: Data could not be decoded!
Parsing object is:
   -    -  Data():
   0 \x00      a: 1 (UBInt16)
   1 \x01
   2 \x00      b: 2 (UBInt16)
   3 \x02
   4 \x00      crc: 0 (CRCB8)
     ^^^^ Checksum mismatch: expected \xfc but found \x00
```

CRCL16

16-bit crc little endian.

```
class oser.CRCL16(strict: bool = False, polynomial: int = 41003, reflect_input: bool = False,
reflect_output: bool = False, initial_value: int = 0, xor_output: int = 0,
automatic_calculation: bool = True, format: str = 'hex')
```

16-bit crc little endian serializer.

Parameters:

- **strict** – bool = False: If set to `True` crc is checked when decoding data. An exception is raised if crcs do not match.
- **polynomial** – int = 0xa02b: the crc polynomial
- **reflect_input** – bool = False: reflect input data
- **reflect_output** – bool = False: reflect output data
- **initial_value** – int = 0x0000: start with *initial_value*

- **xor_output** – int = 0x0000: xor output data with *xor_output*
- **automatic_calculation** – bool = True: if set to `True` crc is automatically calculated when encoding data
- **format** – str = “hex” (string): Use “hex” to format values to hexadecimal. Use “bin” to format values to binary. Use `None` for decimal format.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string into a byte type 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() → 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 (*object*) – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
set(value: int) → None
```

Set the value.

Parameters:
value – the new value

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

Enable or disable automatic calculation when `encode` is called.

Parameters:
enabled=True (*bool*) – the new enabled state.

```
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.

Usage:

```
>>> from oser import ByteStruct, UBInt16, CRCL16, to_hex

>>> class Data(ByteStruct):
...         def __init__(self):
...             super(Data, self).__init__()
...
...             self.a = ULInt16(1)
...             self.b = ULInt16(2)
...
...             self.crc = CRCL16(strict=True)
...
>>> instance = Data()
>>> print(instance)
Data():
    a: 1 (ULInt16)
    b: 2 (ULInt16)
    crc: 0 (CRCL16)

>>> print(instance.introspect())
   -    -  Data():
   0 \x01      a: 1 (ULInt16)
   1 \x00
   2 \x02      b: 2 (ULInt16)
   3 \x00
   4 \x00      crc: 0 (CRCL16)
   5 \x00

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3|  4|  5
\x01\x00\x02\x00\xFC\xC3
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
6
>>> print(instance)
Data():
    a: 1 (ULInt16)
    b: 2 (ULInt16)
    crc: 50172 (CRCL16)

>>> print(instance.introspect())
   -    -  Data():
   0 \x01      a: 1 (ULInt16)
   1 \x00
   2 \x02      b: 2 (ULInt16)
   3 \x00
   4 \xfc      crc: 50172 (CRCL16)
   5 \xc3
```

```
>>> bytes_decoded = instance.decode(b"\x01\x00\x02\x00\xFC\xC3")
>>> print(bytes_decoded)
6
>>> print(instance)
Data():
    a: 1 (ULInt16)
    b: 2 (ULInt16)
    crc: 50172 (CRCL16)

>>> print(instance.introspect())
   -    -  Data():
   0 \x01      a: 1 (ULInt16)
   1 \x00
   2 \x02      b: 2 (ULInt16)
   3 \x00
   4 \xfc      crc: 50172 (CRCL16)
   5 \xc3


>>> bytes_decoded = instance.decode(b"\x01\x00\x02\x00\x00\x00")  # corrupted crc
oser.ChecksumMismatchException: Data could not be decoded!
Parsing object is:
   -    -  Data():
   0 \x01      a: 1 (ULInt16)
   1 \x00
   2 \x02      b: 2 (ULInt16)
   3 \x00
   4 \x00      crc: 0 (CRCL16)
   5 \x00
     ^^^^ Checksum mismatch: expected \xc3fc but found \x0000
```

CRCB16

16-bit crc big endian.

```
class oser.CRCB16(strict: bool = False, polynomial: int = 41003, reflect_input: bool = False,
reflect_output: bool = False, initial_value: int = 0, xor_output: int = 0,
automatic_calculation: bool = True, format: str = 'hex')
```

16-bit crc big endian serializer.

Parameters:

- **strict** – bool = False: If set to `True` crc is checked when decoding data. An exception is raised if crcs do not match.
- **polynomial** – int = 0xa02b: the crc polynomial
- **reflect_input** – bool = False: reflect input data
- **reflect_output** – bool = False: reflect output data
- **initial_value** – int = 0x0000: start with *initial_value*
- **xor_output** – int = 0x0000: xor output data with *xor_output*
- **automatic_calculation** – bool = True: if set to `True` crc is automatically calculated when encoding data
- **format** – str = “hex” (string): Use “hex” to format values to hexadecimal. Use “bin” to format values to binary. Use `None` for decimal format.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string into a byte type 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() → 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 (*object*) – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
set(value: int) → None
```

Set the value.

Parameters:
value – the new value

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

Enable or disable automatic calculation when `encode` is called.

Parameters:
enabled=True (*bool*) – the new enabled state.

```
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.

Usage:

```
>>> from oser import ByteStruct, UBInt16, CRCB16, to_hex

>>> class Data(ByteStruct):
...         def __init__(self):
...             super(Data, self).__init__()
...
...             self.a = UBInt16(1)
...             self.b = UBInt16(2)
...
...             self.crc = CRCB16(strict=True)
...
>>> instance = Data()
>>> print(instance)
Data():
    a: 1 (UBInt16)
    b: 2 (UBInt16)
    crc: 0 (CRCB16)

>>> print(instance.introspect())
   -    -  Data():
   0 \x00      a: 1 (UBInt16)
   1 \x01
   2 \x00      b: 2 (UBInt16)
   3 \x02
   4 \x00      crc: 0 (CRCB16)
   5 \x00

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3|  4|  5
\x00\x01\x00\x02\xF8\x51
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
6
>>> print(instance)
Data():
    a: 1 (UBInt16)
    b: 2 (UBInt16)
    crc: 63569 (CRCB16)

>>> print(instance.introspect())
   -    -  Data():
   0 \x00      a: 1 (UBInt16)
   1 \x01
   2 \x00      b: 2 (UBInt16)
   3 \x02
   4 \xf8      crc: 63569 (CRCB16)
   5 \x51


>>> bytes_decoded = instance.decode(b"\x00\x01\x00\x02\xF8\x51")
>>> print(bytes_decoded)
6
>>> print(instance)
```

```
Data():
    a: 1 (UBInt16)
    b: 2 (UBInt16)
    crc: 63569 (CRCB16)

>>> print(instance.introspect())
   -    -  Data():
   0 \x00      a: 1 (UBInt16)
   1 \x01
   2 \x00      b: 2 (UBInt16)
   3 \x02
   4 \xf8      crc: 63569 (CRCB16)
   5 \x51


>>> bytes_decoded = instance.decode(b"\x00\x01\x00\x02\x00\x00")  # corrupted crc
oser.ChecksumMismatchException: Data could not be decoded!
Parsing object is:
   -    -  Data():
   0 \x00      a: 1 (UBInt16)
   1 \x01
   2 \x00      b: 2 (UBInt16)
   3 \x02
   4 \x00      crc: 0 (CRCB16)
   5 \x00
     ^^^^ Checksum mismatch: expected \xf851 but found \x0000
```

CRCL32

32-bit crc little endian.

```
class oser.CRCL32(strict: bool = False, polynomial: int = 517762881, reflect_input: bool =
False, reflect_output: bool = False, initial_value: int = 0, xor_output: int = 0,
automatic_calculation: bool = True, format: str = 'hex')
```

32-bit crc little endian serializer.

Parameters:

- **strict** – bool = False: If set to `True` crc is checked when decoding data. An exception is raised if crcs do not match.
- **polynomial** – int = 0x1edc6f41: the crc polynomial
- **reflect_input** – bool = False: reflect input data
- **reflect_output** – bool = False: reflect output data
- **initial_value** – int = 0x00000000: start with *initial_value*
- **xor_output** – int = 0x00000000: xor output data with *xor_output*
- **automatic_calculation** – bool = True: if set to `True` crc is automatically calculated when encoding data
- **format** – str = “hex” (string): Use “hex” to format values to hexadecimal. Use “bin” to format values to binary. Use `None` for decimal format.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string into a byte type 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() → 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 (*object*) – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
set(value: int) → None
```

Set the value.

Parameters:
value – the new value

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

Enable or disable automatic calculation when `encode` is called.

Parameters:
enabled=True (*bool*) – the new enabled state.

```
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.

Usage:

```
>>> from oser import ByteStruct, UBInt16, CRCL32, to_hex

>>> class Data(ByteStruct):
...         def __init__(self):
...             super(Data, self).__init__()
...
...             self.a = ULInt16(1)
...             self.b = ULInt16(2)
...
...             self.crc = CRCL32(strict=True)
...
>>> instance = Data()
>>> print(instance)
Data():
    a: 1 (ULInt16)
    b: 2 (ULInt16)
    crc: 0 (CRCL32)

>>> print(instance.introspect())
   -    -  Data():
   0 \x01      a: 1 (ULInt16)
   1 \x00
   2 \x02      b: 2 (ULInt16)
   3 \x00
   4 \x00      crc: 0 (CRCL32)
   5 \x00
   6 \x00
   7 \x00

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3|  4|  5|  6|  7
\x01\x00\x02\x00\xB5\xED\xF0\xDC
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
8
>>> print(instance)
Data():
    a: 1 (ULInt16)
    b: 2 (ULInt16)
    crc: 3706777013 (CRCL32)

>>> print(instance.introspect())
   -    -  Data():
   0 \x01      a: 1 (ULInt16)
   1 \x00
   2 \x02      b: 2 (ULInt16)
   3 \x00
   4 \xb5      crc: 3706777013 (CRCL32)
   5 \xed
   6 \xf0
   7 \xdc
```

```
>>> bytes_decoded = instance.decode(b"\x01\x00\x02\x00\xB5\xED\xF0\xDC")
>>> print(bytes_decoded)
8
>>> print(instance)
Data():
    a: 1 (ULInt16)
    b: 2 (ULInt16)
    crc: 3706777013 (CRCL32)

>>> print(instance.introspect())
   -    -  Data():
   0 \x01      a: 1 (ULInt16)
   1 \x00
   2 \x02      b: 2 (ULInt16)
   3 \x00
   4 \xb5      crc: 3706777013 (CRCL32)
   5 \xed
   6 \xf0
   7 \xdc


>>> bytes_decoded = instance.decode(b"\x01\x00\x02\x00\x00\x00\x00\x00")  # corrupted crc
oser.ChecksumMismatchException: Data could not be decoded!
Parsing object is:
   -    -  Data():
   0 \x01      a: 1 (ULInt16)
   1 \x00
   2 \x02      b: 2 (ULInt16)
   3 \x00
   4 \x00      crc: 0 (CRCL32)
   5 \x00
   6 \x00
   7 \x00
     ^^^^ Checksum mismatch: expected \xdcf0edb5 but found \x00000000
```

CRCB32

32-bit crc big endian.

```
class oser.CRCB32(strict: bool = False, polynomial: int = 517762881, reflect_input: bool =
False, reflect_output: bool = False, initial_value: int = 0, xor_output: int = 0,
automatic_calculation: bool = True, format: str = 'hex')
```

32-bit crc big endian serializer.

Parameters:

- **strict** – bool = False: If set to `True` crc is checked when decoding data. An exception is raised if crcs do not match.
- **polynomial** – int = 0x1edc6f41: the crc polynomial
- **reflect_input** – bool = False: reflect input data
- **reflect_output** – bool = False: reflect output data
- **initial_value** – int = 0x00000000: start with *initial_value*
- **xor_output** – int = 0x00000000: xor output data with *xor_output*

- **automatic_calculation** – bool = True: if set to `True` crc is automatically calculated when encoding data
- **format** – str = “hex” (string): Use “hex” to format values to hexadecimal. Use “bin” to format values to binary. Use `None` for decimal format.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string into a byte type 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() → 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 (*object*) – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
set(value: int) → None
```

Set the value.

Parameters:
value – the new value

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

Enable or disable automatic calculation when `encode` is called.

Parameters:
enabled=True (*bool*) – the new enabled state.

```
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.

Usage:

```
>>> from oser import ByteStruct, UBInt16, CRCB32, to_hex

>>> class Data(ByteStruct):
...         def __init__(self):
...             super(Data, self).__init__()
...
...             self.a = UBInt16(1)
...             self.b = UBInt16(2)
...
...             self.crc = CRCB32(strict=True)
...
>>> instance = Data()
>>> print(instance)
Data():
    a: 1 (UBInt16)
    b: 2 (UBInt16)
    crc: 0 (CRCB32)

>>> print(instance.introspect())
   -    -  Data():
   0 \x00      a: 1 (UBInt16)
   1 \x01
   2 \x00      b: 2 (UBInt16)
   3 \x02
   4 \x00      crc: 0 (CRCB32)
   5 \x00
   6 \x00
   7 \x00

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3|  4|  5|  6|  7
\x00\x01\x00\x02\xFF\x56\x3A\x53
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
8
>>> print(instance)
Data():
    a: 1 (UBInt16)
    b: 2 (UBInt16)
    crc: 4283841107 (CRCB32)

>>> print(instance.introspect())
   -    -  Data():
   0 \x00      a: 1 (UBInt16)
   1 \x01
   2 \x00      b: 2 (UBInt16)
   3 \x02
   4 \xff      crc: 4283841107 (CRCB32)
   5 \x56
```

```
   6 \x3a
   7 \x53


>>> bytes_decoded = instance.decode(b"\x00\x01\x00\x02\xFF\x56\x3A\x53")
>>> print(bytes_decoded)
8
>>> print(instance)
Data():
    a: 1 (UBInt16)
    b: 2 (UBInt16)
    crc: 4283841107 (CRCB32)

>>> print(instance.introspect())
   -    -  Data():
   0 \x00      a: 1 (UBInt16)
   1 \x01
   2 \x00      b: 2 (UBInt16)
   3 \x02
   4 \xff      crc: 4283841107 (CRCB32)
   5 \x56
   6 \x3a
   7 \x53


>>> bytes_decoded = instance.decode(b"\x00\x01\x00\x02\x00\x00\x00\x00")  # corrupted crc
oser.ChecksumMismatchException: Data could not be decoded!
Parsing object is:
   -    -  Data():
   0 \x00      a: 1 (UBInt16)
   1 \x01
   2 \x00      b: 2 (UBInt16)
   3 \x02
   4 \x00      crc: 0 (CRCB32)
   5 \x00
   6 \x00
   7 \x00
     ^^^^ Checksum mismatch: expected \xff563a53 but found \x00000000
```

CRCL64

64-bit crc little endian.

```
class oser.CRCL64(strict: bool = False, polynomial: int = 4823603603198064275, reflect_input:
bool = False, reflect_output: bool = False, initial_value: int = 0, xor_output: int = 0,
automatic_calculation: bool = True, format: str = 'hex')
```

64-bit crc little endian serializer.

Parameters:

- **strict** – bool = False: If set to `True` crc is checked when decoding data. An exception is raised if crcs do not match.
- **polynomial** – int = 0x42f0e1eba9ea3693: the crc polynomial
- **reflect_input** – bool = False: reflect input data

- **reflect_output** – bool = False: reflect output data
- **initial_value** – int = 0x0000000000000000: start with *initial_value*
- **xor_output** – int = 0x0000000000000000: xor output data with *xor_output*
- **automatic_calculation** – bool = True: if set to `True` crc is automatically calculated when encoding data
- **format** – str = "hex" (string): Use "hex" to format values to hexadecimal. Use "bin" to format values to binary. Use `None` for decimal format.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string into a byte type 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() → 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 (*object*) – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
set(value: int) → None
```

Set the value.

Parameters:
value – the new value

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

Enable or disable automatic calculation when `encode` is called.

Parameters:
enabled=True (*bool*) – the new enabled state.

```
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.

Usage:

```
>>> from oser import ByteStruct, UBInt16, CRCL64, to_hex

>>> class Data(ByteStruct):
...         def __init__(self):
...             super(Data, self).__init__()
...
...             self.a = ULInt16(1)
...             self.b = ULInt16(2)
...
...             self.crc = CRCL64(strict=True)
...
>>> instance = Data()
>>> print(instance)
Data():
    a: 1 (ULInt16)
    b: 2 (ULInt16)
    crc: 0 (CRCL64)

>>> print(instance.introspect())
   -    -  Data():
   0 \x01      a: 1 (ULInt16)
   1 \x00
   2 \x02      b: 2 (ULInt16)
   3 \x00
   4 \x00      crc: 0 (CRCL64)
   5 \x00
   6 \x00
   7 \x00
   8 \x00
   9 \x00
  10 \x00
  11 \x00

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11
\x01\x00\x02\x00\xEA\x1F\x12\x02\xC4\xF9\x66\xF9
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
12
>>> print(instance)
Data():
    a: 1 (ULInt16)
    b: 2 (ULInt16)
    crc: 17971325983312191466 (CRCL64)

>>> print(instance.introspect())
```

```
   -    -  Data():
   0 \x01      a: 1 (ULInt16)
   1 \x00
   2 \x02      b: 2 (ULInt16)
   3 \x00
   4 \xea      crc: 17971325983312191466 (CRCL64)
   5 \x1f
   6 \x12
   7 \x02
   8 \xc4
   9 \xf9
  10 \x66
  11 \xf9


>>> bytes_decoded = instance.decode(b"\x01\x00\x02\x00\xEA\x1F\x12\x02\xC4\xF9\x66\xF9")
>>> print(bytes_decoded)
12
>>> print(instance)
Data():
    a: 1 (ULInt16)
    b: 2 (ULInt16)
    crc: 17971325983312191466 (CRCL64)

>>> print(instance.introspect())
   -    -  Data():
   0 \x01      a: 1 (ULInt16)
   1 \x00
   2 \x02      b: 2 (ULInt16)
   3 \x00
   4 \xea      crc: 17971325983312191466 (CRCL64)
   5 \x1f
   6 \x12
   7 \x02
   8 \xc4
   9 \xf9
  10 \x66
  11 \xf9


>>> bytes_decoded = instance.decode(b"\x01\x00\x02\x00\x00\x00\x00\x00\x00\x00\x00\x00")  #
corrupted crc
oser.ChecksumMismatchException: Data could not be decoded!
Parsing object is:
   -    -  Data():
   0 \x01      a: 1 (ULInt16)
   1 \x00
   2 \x02      b: 2 (ULInt16)
   3 \x00
   4 \x00      crc: 0 (CRCL64)
   5 \x00
   6 \x00
   7 \x00
   8 \x00
   9 \x00
  10 \x00
  11 \x00
     ^^^^ Checksum mismatch: expected \xf966f9c402121fea but found \x0000000000000000
```

CRCB64

64-bit crc big endian.

```
class oser.CRCB64(strict: bool = False, polynomial: int = 4823603603198064275, reflect_input:
bool = False, reflect_output: bool = False, initial_value: int = 0, xor_output: int = 0,
automatic_calculation: bool = True, format: str = 'hex')
```

64-bit crc big endian serializer.

Parameters:

- **strict** – bool = False: If set to `True` crc is checked when decoding data. An exception is raised if crcs do not match.
- **polynomial** – int = 0x42f0e1eba9ea3693: the crc polynomial
- **reflect_input** – bool = False: reflect input data
- **reflect_output** – bool = False: reflect output data
- **initial_value** – int = 0x0000000000000000: start with *initial_value*
- **xor_output** – int = 0x0000000000000000: xor output data with *xor_output*
- **automatic_calculation** – bool = True: if set to `True` crc is automatically calculated when encoding data
- **format** – str = “hex” (string): Use “hex” to format values to hexadecimal. Use “bin” to format values to binary. Use `None` for decimal format.

```
decode(data: bytes, full_data: bytes = b'', context_data: bytes = b'') → int
```

Decode a binary string into a byte type 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() → 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 (*object*) – stop introspection at `stop_at`.

```
root() → ByteStruct | BitStruct
```

return root element

```
set(value: int) → None
```

Set the value.

Parameters:
value – the new value

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

Enable or disable automatic calculation when `encode` is called.

Parameters:
enabled=True (*bool*) – the new enabled state.

```
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.

Usage:

```
>>> from oser import ByteStruct, UBInt16, CRCB64, to_hex

>>> class Data(ByteStruct):
...         def __init__(self):
...             super(Data, self).__init__()
...
...             self.a = UBInt16(1)
...             self.b = UBInt16(2)
...
...             self.crc = CRCB64(strict=True)
...
>>> instance = Data()
>>> print(instance)
Data():
    a: 1 (UBInt16)
    b: 2 (UBInt16)
    crc: 0 (CRCB64)

>>> print(instance.introspect())
   -    -  Data():
   0 \x00      a: 1 (UBInt16)
   1 \x01
   2 \x00      b: 2 (UBInt16)
   3 \x02
   4 \x00      crc: 0 (CRCB64)
   5 \x00
```

```
   6 \x00
   7 \x00
   8 \x00
   9 \x00
  10 \x00
  11 \x00

>>> binary = instance.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11
\x00\x01\x00\x02\xA6\x0F\x34\x48\x69\x03\x75\xE1
>>> bytes_decoded = instance.decode(binary)
>>> print(bytes_decoded)
12
>>> print(instance)
Data():
    a: 1 (UBInt16)
    b: 2 (UBInt16)
    crc: 11965840220550821345 (CRCB64)

>>> print(instance.introspect())
   -    -  Data():
   0 \x00      a: 1 (UBInt16)
   1 \x01
   2 \x00      b: 2 (UBInt16)
   3 \x02
   4 \xa6      crc: 11965840220550821345 (CRCB64)
   5 \x0f
   6 \x34
   7 \x48
   8 \x69
   9 \x03
  10 \x75
  11 \xe1


>>> bytes_decoded = instance.decode(b"\x00\x01\x00\x02\xA6\x0F\x34\x48\x69\x03\x75\xE1")
>>> print(bytes_decoded)
12
>>> print(instance)
Data():
    a: 1 (UBInt16)
    b: 2 (UBInt16)
    crc: 11965840220550821345 (CRCB64)

>>> print(instance.introspect())
   -    -  Data():
   0 \x00      a: 1 (UBInt16)
   1 \x01
   2 \x00      b: 2 (UBInt16)
   3 \x02
   4 \xa6      crc: 11965840220550821345 (CRCB64)
   5 \x0f
   6 \x34
   7 \x48
   8 \x69
   9 \x03
  10 \x75
  11 \xe1
```

```
>>> bytes_decoded = instance.decode(b"\x00\x01\x00\x02\x00\x00\x00\x00\x00\x00\x00\x00") # corrupted crc
oser.ChecksumMismatchException: Data could not be decoded!
Parsing object is:
   -    -  Data():
   0 \x00      a: 1 (UBInt16)
   1 \x01
   2 \x00      b: 2 (UBInt16)
   3 \x02
   4 \x00      crc: 0 (CRCB64)
   5 \x00
   6 \x00
   7 \x00
   8 \x00
   9 \x00
  10 \x00
  11 \x00
     ^^^^ Checksum mismatch: expected \xa60f3448690375e1 but found \x0000000000000000
```

Special Types

Constant

You can make any value a constant using `oser.Constant()`.

`oser.Constant()` raises a *ValueError* if the current value does not match the desired value.

```
oser.Constant(*args: ByteType | BitType, **kwargs: Any) → ByteType | BitType
```

Wrapper to make any OSER type or class a constant.

Parameters:

- ***args** (*tuple*) – the first positional argument is the class that will be made constant. All other parameters are passed to the __init__ of the first positional argument.
- ****kwargs** – keyword parameters are passed to the __init__ of the first positional argument.

Usage:

```
>>> from oser import ByteStruct, Constant, ULInt8, String, to_hex

>>> c = Constant(ULInt8, 23)  # constant scalar
>>> print(c.introspect())
   0 \x17  23 (ULInt8)

>>> binary = c.encode()
>>> print(to_hex(binary))
   0
\x17
>>> c.decode(b"\x00")
```

```
ValueError: Constant value not matched: expected value: 23, decoded value: 0

>>> class Foo(ByteStruct):  # constant scalar nested in a ByteStruct
...     def __init__(self):
...         super(Foo, self).__init__()
...         self.c = Constant(ULInt8, 23)
...
>>> f = Foo()
>>> print(f.introspect())
   -    -  Foo():
   0 \x17      c: 23 (ULInt8)

>>> binary = f.encode()
>>> print(to_hex(binary))
   0
\x17
>>> f.decode(b"\x00")
ValueError: Constant value not matched: expected value: 23, decoded value: 0

>>> c = Constant(String, length=10, value=b"abcdefghij")  # constant String
>>> print(c.introspect())
   -    -  String():
   0 \x61      'a'
   1 \x62      'b'
   2 \x63      'c'
   3 \x64      'd'
   4 \x65      'e'
   5 \x66      'f'
   6 \x67      'g'
   7 \x68      'h'
   8 \x69      'i'
   9 \x6a      'j'

>>> binary = c.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9
\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6A
>>> c.decode(b"\x61\x62\x63\x64\x65\x66\x67\x68\x69B")
ValueError: Constant value not matched: expected value: abcdefghij, decoded value: abcdefghiB
```

Google Protocol Buffers Adapter

Google Protocol Buffers can be used with an adapter. This way it is possible to create protocols that include messages using Google Protocol Buffers but can also be extended with a CRC for example.

```
class oser.ProtocolBuffersAdapter(instance: Any | None = None, length: Callable | int |
None = None)
```

`ProtocolBuffersAdapter` builds a Google Protocol Buffers adapter.

Parameters:

- **instance=None** – an instance of a Google Protocol Buffers class.
- **length=None** – the length of the serialized data. If `None` the rest of the data is used so in most cases it should be a lambda expression. Look at the examples for more details.

```
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.

If `length` is None, decoding is done iterative starting with the full buffer and removing one last byte after another until the buffer can be decoded. The buffer must fit the decoded message in protobuf and must not be longer or shorter.

If decoding is not possible at all a `DecodeException` is raised.

It is recommended not to set `length` to `None` even if it works.

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() → Any | None
```

Return the protocol buffers instance.

```
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(instance: Any | None) → None
```

Set the protocol buffers instance.

Parameters:
instance – the protocol buffers instance

```
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.

The example uses the following proto file:

```
package tutorial;

message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }

  repeated PhoneNumber phone = 4;
}

message AddressBook {
  repeated Person person = 1;
}
```

Usage:

```
>>> from oser import ByteStruct, ProtocolBuffersAdapter, UBInt16, CRCB32, to_hex
>>> from tutorial_pb2 import Person

>>> class Data(ByteStruct):
...         def __init__(self):
...             super(Data, self).__init__()
...
...             self.payloadLength = UBInt16(0)
...             person = Person()
...             person.id = 1
...             person.name = "name"
...             person.PhoneNumber.number = 2
...             self.payload = ProtocolBuffersAdapter(object=person, length=lambda self:
self.payloadLength.get())
...             self.crc = CRCB32(strict=True)
...
...         def encode(self, full_data="", context_data=""):#
...             self.payloadLength.set(len(self.payload.encode()))
...             #print("set length to", self.length.get())
...             return super(Data, self).encode()
...
>>> instance = Data()

>>> binary = instance.encode()
>>> print(instance)
Data():
    payloadLength: 8 (UBInt16)
```

```
    payload:
\x04name\x10\x01
    crc: 3809365856 (CRCB32)

>>> print(instance.introspect())
   -    -  Data():
   0 \x00      payloadLength: 8 (UBInt16)
   1 \x08
   -    -      payload: ProtocolBuffersAdapter(Person):
   2 \x0a          '\n'
   3 \x04          '\x04'
   4 \x6e          'n'
   5 \x61          'a'
   6 \x6d          'm'
   7 \x65          'e'
   8 \x10          '\x10'
   9 \x01          '\x01'
  10 \xe3      crc: 3809365856 (CRCB32)
  11 \x0e
  12 \x4f
  13 \x60

>>> print(to_hex(binary))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12| 13
\x00\x08\x0A\x04\x6E\x61\x6D\x65\x10\x01\xE3\x0E\x4F\x60

>>> # alter data
... person = instance.payload.get()
>>> person.id = 123456789
>>> person.name = "another name"
>>> person.PhoneNumber.number = 1337

>>> binary = instance.encode()
>>> print(instance)
Data():
    payloadLength: 19 (UBInt16)
    payload:

another name\x10���:
    crc: 3147422169 (CRCB32)

>>> print(instance.introspect())
   -    -  Data():
   0 \x00      payloadLength: 19 (UBInt16)
   1 \x13
   -    -      payload: ProtocolBuffersAdapter(Person):
   2 \x0a          '\n'
   3 \x0c          '\x0c'
   4 \x61          'a'
   5 \x6e          'n'
   6 \x6f          'o'
   7 \x74          't'
   8 \x68          'h'
   9 \x65          'e'
  10 \x72          'r'
  11 \x20          ' '
  12 \x6e          'n'
  13 \x61          'a'
  14 \x6d          'm'
  15 \x65          'e'
```

```
  16 \x10          '\x10'
  17 \x95          '\x95'
  18 \x9a          '\x9a'
  19 \xef          '\xef'
  20 \x3a          ':'
  21 \xbb      crc: 3147422169 (CRCB32)
  22 \x99
  23 \xd9
  24 \xd9

>>> 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
\x00\x13\x0A\x0C\x61\x6E\x6F\x74\x68\x65\x72\x20\x6E\x61\x6D\x65\x10\x95\x9A\xEF\x3A\xBB\x99\xD9\xD9

>>> bytes_decoded =
instance.decode(b"\x00\x10\x0A\x0C\x48\x49\x4C\x53\x54\x45\x52\x20\x47\x6D\x62\x48\x10\x01\x0F\x5B\x
58\x85")
>>> print(bytes_decoded)
22
>>> print(instance)
Data():
    payloadLength: 16 (UBInt16)
    payload:

HILSTER GmbH\x10\x01
    crc: 257644677 (CRCB32)

>>> print(instance.introspect())
   -    -  Data():
   0 \x00      payloadLength: 16 (UBInt16)
   1 \x10
   -    -      payload: ProtocolBuffersAdapter(Person):
   2 \x0a          '\n'
   3 \x0c          '\x0c'
   4 \x48          'H'
   5 \x49          'I'
   6 \x4c          'L'
   7 \x53          'S'
   8 \x54          'T'
   9 \x45          'E'
  10 \x52          'R'
  11 \x20          ' '
  12 \x47          'G'
  13 \x6d          'm'
  14 \x62          'b'
  15 \x48          'H'
  16 \x10          '\x10'
  17 \x01          '\x01'
  18 \x0f      crc: 257644677 (CRCB32)
  19 \x5b
  20 \x58
  21 \x85


>>> print("name:", instance.payload.get().name)
name: HILSTER GmbH
```

JSON Adapter

It is possible to serialize and deserialize JSON with OSER.

```
class oser.JSONAdapter(data: Any)
```

`JSONAdapter` builds a JSON serializer.

Parameters:
data=None – the data to be json-ized.

```
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() → Any
```

Return the value.

Returns:
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: 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 – 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.

Usage:

```
>>> from oser import JSONAdapter

>>> data = {
...     "list" : [ii for ii in range(3)],
...     "scalar" : 1337,
...     "string" : "test"
...     }

>>> j = JSONAdapter(data=data)
>>> print(j)
{'scalar': 1337, 'list': [0, 1, 2], 'string': 'test'}

>>> print(j.encode())
{"list":[0,1,2],"scalar":1337,"string":"test"}
>>> print(j.introspect())
   -    -  JSONAdapter():
   0 \x7b      '{'
   1 \x22      '"'
   2 \x6c      'l'
   3 \x69      'i'
   4 \x73      's'
   5 \x74      't'
   6 \x22      '"'
   7 \x3a      ':'
   8 \x5b      '['
   9 \x30      '0'
  10 \x2c      ','
  11 \x31      '1'
  12 \x2c      ','
  13 \x32      '2'
  14 \x5d      ']'
  15 \x2c      ','
  16 \x22      '"'
  17 \x73      's'
  18 \x63      'c'
  19 \x61      'a'
  20 \x6c      'l'
  21 \x61      'a'
  22 \x72      'r'
  23 \x22      '"'
  24 \x3a      ':'
  25 \x31      '1'
  26 \x33      '3'
  27 \x33      '3'
  28 \x37      '7'
  29 \x2c      ','
  30 \x22      '"'
  31 \x73      's'
  32 \x74      't'
  33 \x72      'r'
```

```
  34 \x69      'i'
  35 \x6e      'n'
  36 \x67      'g'
  37 \x22      '"'
  38 \x3a      ':'
  39 \x22      '"'
  40 \x74      't'
  41 \x65      'e'
  42 \x73      's'
  43 \x74      't'
  44 \x22      '"'
  45 \x7d      '}'
  46 \x00      '\x00'

>>> j2 = JSONAdapter()
>>> j2.decode(b"[1,2,3]\x00")
8
>>> print(j2)
[1, 2, 3]

>>> print(j2.get())
[1, 2, 3]
>>> print(type(j2.get()))
<type 'list'>
>>> j3 = JSONAdapter()
>>> j3.set([4,5,6])
>>> print(j3)
```

Regular Expression Match

If you need a `oser.String` that fulfills some constraints you can use `oser.RegularExpressionMatch()` to do this. `oser.RegularExpressionMatch()` expects a regular expression that is applied to the *String`s value. A `RegularExpressionMismatchException* is raised if the current value does not match the regular expression.

```
oser.RegularExpressionMatch(pattern: str | bytes, length: Callable | int | None = None,
value: bytes | str = b'', padding: bytes | str = b'\x00') → String
```

`RegularExpressionMatch` creates a `oser.String` matches a regular expression.

Parameters:

- **pattern** – a regular expression pattern.
- **length** – the length of the underlying string.
- **value=""** – the initial value.
- **padding=b"\x00"** – padding character for the string when it is encoded.

Usage (only match numeric strings):

```
>>> from oser import ByteStruct, ULInt8, RegularExpressionMatch, to_hex

>>> class Foo(ByteStruct):
```

```
...     def __init__(self):
...         super(Foo, self).__init__()
...
...         self.length = ULInt8(10)
...         self.numbers = RegularExpressionMatch(pattern=r"^[0-9]*$", length=lambda self:
self.length.get(),
                value=b"0123456789")
...
>>> f = Foo()
>>> print(f.introspect())
   -    -  Foo():
   0 \x0a      length: 10 (ULInt8)
   -    -      numbers: RegularExpressionMatch():
   1 \x30          '0'
   2 \x31          '1'
   3 \x32          '2'
   4 \x33          '3'
   5 \x34          '4'
   6 \x35          '5'
   7 \x36          '6'
   8 \x37          '7'
   9 \x38          '8'
  10 \x39          '9'

>>> binary = f.encode()
>>> print(to_hex(binary))
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10
\x0A\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39
>>> f.decode(b"\x0A\x30\x31\x32\x33\x34\x35\x36\x37\x38A")
oser.RegularExpressionMismatchException: Current value (012345678A) does not match regular
expression (^[0-9]*$)
```

Repeat Until

Sometimes you have optional data in your data stream or you need to decode items until the end of a stream or until a stop condition is met. Then `oser.Array` might not be an option if you don't know the number of items.

In this cases `oser.RepeatUntil` can help. In general it works like `oser.Array` but `length` is not a callable. For encoding you can set the items and the length and for decoding the length is determined dynamically while `oser.RepeatUntil` consumes items until `stop_condition` returns `True`, an exception is raised or the data stream ends.

```
class oser.RepeatUntil(length: int, stop_condition: Callable | bool | None, prototype:
ByteStruct | BitStruct | ByteType | BitType, stop_on_exception: bool = False, args: Tuple |
None = None, kwargs: Dict | None = None, values: List[ByteStruct | BitStruct | ByteType |
BitType] | None = None)
```

`RepeatUntil` builds an array serializer with a variable length that is determined dynamically on decoding.

Parameters:

- **length** – states the initial length

- **stop_condition** – the stop condition called with `context`, `data` (the next bytes to be parsed), `full_data` (all data) and `context_data` the data within the current hierarchy. If `stop_condition` returns `True` decoding stops.
- **prototype** – a class that is the prototype for *values*
- **stop_on_exception** – if set to `True` a `DecodeException` is handled as a stop condition.
- **args=None** – a tuple of positional arguments that are passed to *prototype* when it is instantiated when array size is extended. Note: don't forget a leading comma when passing one value `(1,)` (`(1)` is an integer!).
- **kwargs=None** – a dictionary of named arguments that are passed to *prototype* when it is instantiated when array size is extended.
- **values=None** – the initial array values. Can be left empty. If empty the array is automatically resized with instances of `prototype(*args, **kwargs)`.

```
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

```
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() → RepeatUntil
```

Return the value.

```
get_byte_size() → int
```

Return the current size in bytes.

```
get_size() → int
```

Return the current 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_length(length: int) → None
```

Set the length.

Parameters:
length – the new length

```
to_dict() → List[Dict[Any, Any]]
```

Return self as a dict()

```
up() → ByteStruct | BitStruct
```

return parent element

Usage:

```
>>> from oser import ByteStruct, RepeatUntil, CRCB32, UBInt8, to_hex

>>> length = 3
>>> class Data(ByteStruct):
...     def __init__(self):
...         super(Data, self).__init__()
...         self.items = RepeatUntil(length,
...                         stop_condition=lambda ctx,
...                             data, full_data,
...                             context_data: len(data) <= 4,
...                         prototype=UBInt8,
...                         values=[UBInt8(ii) for ii in range(length)])
...         self.crc = CRCB32(strict=True)
...
>>> data = Data()
>>> data.items.set_length(20)  # resize
>>> for ii in range(20):
...     data.items[ii].set(ii)
...
>>> encoded = data.encode()
>>> print(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
\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0A\x0B\x0C\x0D\x0E\x0F\x10\x11\x12\x13\x6D\xA9\x2B\x61
>>> print(data.introspect())
   -    -  Data():
   -    -      items: RepeatUntil():
   -    -      [
   0 \x00          @0: 0 (UBInt8)
   1 \x01          @1: 1 (UBInt8)
   2 \x02          @2: 2 (UBInt8)
   3 \x03          @3: 3 (UBInt8)
   4 \x04          @4: 4 (UBInt8)
   5 \x05          @5: 5 (UBInt8)
   6 \x06          @6: 6 (UBInt8)
   7 \x07          @7: 7 (UBInt8)
   8 \x08          @8: 8 (UBInt8)
   9 \x09          @9: 9 (UBInt8)
  10 \x0a          @10: 10 (UBInt8)
  11 \x0b          @11: 11 (UBInt8)
  12 \x0c          @12: 12 (UBInt8)
  13 \x0d          @13: 13 (UBInt8)
  14 \x0e          @14: 14 (UBInt8)
  15 \x0f          @15: 15 (UBInt8)
  16 \x10          @16: 16 (UBInt8)
```

```
  17 \x11          @17: 17 (UBInt8)
  18 \x12          @18: 18 (UBInt8)
  19 \x13          @19: 19 (UBInt8)
   -    -       ]
  20 \x6d      crc: 1839803233 (CRCB32)
  21 \xa9
  22 \x2b
  23 \x61

>>> b = b"\x00\x01\x02\x03\xC1\xE9\xBD\xED"
>>> data2 = Data()
>>> decoded = data2.decode(b)
>>> print(data2.introspect())
   -    -  Data():
   -    -      items: RepeatUntil():
   -    -      [
   0 \x00          @0: 0 (UBInt8)
   1 \x01          @1: 1 (UBInt8)
   2 \x02          @2: 2 (UBInt8)
   3 \x03          @3: 3 (UBInt8)
   -    -      ]
   4 \xc1      crc: 3253321197 (CRCB32)
   5 \xe9
   6 \xbd
   7 \xed
```

Lazy Initialization

Sometimes you need to speed up some things. You can use the `oser.Lazy` element to initialize an object on first demand. So if an object is never used it is never initialized saving memory and cpu time.

```
class oser.Lazy(cls: Type[ByteStruct | BitStruct | ByteType | BitType], *args: ByteStruct |
BitStruct | ByteType | BitType, **kwargs: Any)
```

`Lazy` can be used to realize lazy-initialization automatically.

cls is instantiated on first access but not in `__init__()`. So if no access is done it will never be instantiated and will not occupy any memory and cpu time.

Parameters:

- **cls** – the class to be instantiated.
- ***args** – a tuple for positional arguments for `cls`.
- ****kwargs** – a dict for named arguments for `cls`.

Usage:

```
>>> from oser import ByteStruct, ULInt8, Lazy

>>> class Foo(ByteStruct):
...     def __init__(self):
...         super(Foo, self).__init__()
```

```
...
...         self.a = Lazy(ULInt8, 2)
...
>>> foo = Foo()
>>> print(foo) # foo.a is now initialized
Foo():
    a: 2 (ULInt8)
```

BitMap

`oser.BitMap` is a bit type that allows you to easily add bits at a given position instead of calculating these positions manually. This makes it easier to implement a protocol with many status bits, etc.

Indexes can either be integers or tuples of byte and bit positions. Both types can be mixed. Every undefined bit position is filled with a `oser.PaddingFlag` that might by strict if the `strict` option is set to `True`.

Every name is converted to lowercase and every character that cannot be part of a class member is removed. Thus it is possible that the names are not unique. If names are not unique they are postfixed with the byte and the bit position. The postfixed members must be used to modify and read values.

Reserved bits are hidden when the instance is printed and are visible in introspection.

```
class oser.BitMap(length: int, values: Dict[int | Tuple[int, int], str] | None = None,
strict: bool = False)
```

BitMap type that automatically generates `oser.PaddingFlag()` at undefined bit positions. Reserved bits are hidden in stringified output.

Parameters:

- **length** (*int*) – the length in bits
- **values=None** (*dict*) – dictionary with key: value where key is the bit index as integer or a tuple of (byte, bit)
- **strict=False** (*bool*) – When True, non-specified values must be 0

```
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.

```
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.

```
from_dict(data: Dict) → None
```

Fill self with data.

Parameters:
data (*dict*) – 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 (*bool*) – 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 bits.

Returns:
the length of the bit type in bits.
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 (*object*) – 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

Usage:

```
>>> import oser

>>> class StructWithIntegerIndexes(oser.BitStruct):
...     def __init__(self):
...         super(StructWithIntegerIndexes, self).__init__()
...
...         self.bitmap = oser.BitMap(8,
...                                   values={
...                                       0: "a",
...                                       1: "b",
...                                       2: "c"
...                                   })
...

>>> s = StructWithIntegerIndexes()
>>> s.bitmap.a.set(True)
>>> print(s)
StructWithIntegerIndexes():
    bitmap: BitMap():
        c: 0 (Flag)
        b: 0 (Flag)
        a: True (Flag)

>>> print(s.introspect())
   -    -  StructWithIntegerIndexes():
   -    -      bitmap: BitMap():
```

```
   0.7  0          reserved_0_7: False (PaddingFlag)
   0.6  0          reserved_0_6: False (PaddingFlag)
   0.5  0          reserved_0_5: False (PaddingFlag)
   0.4  0          reserved_0_4: False (PaddingFlag)
   0.3  0          reserved_0_3: False (PaddingFlag)
   0.2  0          c: 0 (Flag)
   0.1  0          b: 0 (Flag)
   0.0  1          a: True (Flag)



>>> class StructWithTupleIndexes(oser.BitStruct):
...     def __init__(self):
...         super(StructWithTupleIndexes, self).__init__()
...
...         self.bitmap = oser.BitMap(8,
...                                   values={
...                                       (0, 0): "a",
...                                       (0, 1): "b",
...                                       (0, 2): "c"
...                                   })
...

>>> s = StructWithTupleIndexes()
>>> s.bitmap.a.set(True)
>>> print(s)
StructWithTupleIndexes():
    bitmap: BitMap():
        c: 0 (Flag)
        b: 0 (Flag)
        a: True (Flag)

>>> print(s.introspect())
   -    -  StructWithTupleIndexes():
   -    -      bitmap: BitMap():
   0.7  0          reserved_0_7: False (PaddingFlag)
   0.6  0          reserved_0_6: False (PaddingFlag)
   0.5  0          reserved_0_5: False (PaddingFlag)
   0.4  0          reserved_0_4: False (PaddingFlag)
   0.3  0          reserved_0_3: False (PaddingFlag)
   0.2  0          c: 0 (Flag)
   0.1  0          b: 0 (Flag)
   0.0  1          a: True (Flag)



>>> class StructWithNonUniqueNames(oser.BitStruct):
...     def __init__(self):
...         super(StructWithNonUniqueNames, self).__init__()
...
...         self.bitmap = oser.BitMap(8,
...                                   values={
...                                       0: "foobar",
...                                       1: "foobar",
...                                   })
...

>>> s = StructWithNonUniqueNames()
>>> print(s)
StructWithNonUniqueNames():
```

```
    bitmap: BitMap():
        foobar_0_1: 0 (Flag)
        foobar_0_0: 0 (Flag)

>>> print(s.introspect())
   -    -  StructWithNonUniqueNames():
   -    -      bitmap: BitMap():
   0.7  0          reserved_0_7: False (PaddingFlag)
   0.6  0          reserved_0_6: False (PaddingFlag)
   0.5  0          reserved_0_5: False (PaddingFlag)
   0.4  0          reserved_0_4: False (PaddingFlag)
   0.3  0          reserved_0_3: False (PaddingFlag)
   0.2  0          reserved_0_2: False (PaddingFlag)
   0.1  0          foobar_0_1: 0 (Flag)
   0.0  0          foobar_0_0: 0 (Flag)
```

Modbus Types

`oser` includes special integer types for modbus applications.

UInt2143

```
class oser.UInt2143(value: int = 0, format: str | None = None)
```

Unsigned, 32-Bit integer with modbus endianness: little endian ordered registers with big endian content.

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.

SInt2143

```
class oser.SInt2143(value: int = 0, format: str | None = None)
```

Signed, 32-Bit integer with modbus endianness: little endian ordered registers with big endian content.

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.

UInt3412

```
class oser.UInt3412(value: int = 0, format: str | None = None)
```

Unsigned, 32-Bit integer with modbus endianness: big endian ordered registers with little endian content.

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.

SInt3412

```
class oser.SInt3412(value: int = 0, format: str | None = None)
```

Signed, 32-Bit integer with modbus endianness: big endian ordered registers with little endian content.

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.

UInt21436587

```
class oser.UInt21436587(value: int = 0, format: str | None = None)
```

Unsigned, 64-Bit integer with modbus endianness: little endian ordered registers with big endian content.

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.

SInt21436587

```
class oser.SInt21436587(value: int = 0, format: str | None = None)
```

Signed, 64-Bit integer with modbus endianness: little endian ordered registers with big endian content.

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.

UInt78563412

```
class oser.UInt78563412(value: int = 0, format: str | None = None)
```

Unsigned, 64-Bit integer with modbus endianness: big endian ordered registers with little endian content.

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.

SInt78563412

```
class oser.SInt78563412(value: int = 0, format: str | None = None)
```

Signed, 64-Bit integer with modbus endianness: big endian ordered registers with little endian content.

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.

Float3412

```
class oser.Float3412(value: float = 0.0)
```

Unsigned, 32-Bit float with modbus endianness: big endian ordered registers with little endian content.

Parameters:
value – float = 0.0: the initial value.

```
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() → float
```

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: float) → 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.

Float2143

```
class oser.Float2143(value: float = 0.0)
```

Unsigned, 32-Bit float with modbus endianness: little endian ordered registers with big endian content.

Parameters:
value – float = 0.0: the initial value.

```
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() → float
```

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: float) → 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.

Double78563412

```
class oser.Double78563412(value: float = 0.0)
```

Unsigned, 64-Bit double with modbus endianness: big endian ordered registers with little endian content.

Parameters:
value – float = 0.0: the initial value.

```
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() → float
```

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: float) → 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.

Double21436587

```
class oser.Double21436587(value: float = 0.0)
```

Unsigned, 64-Bit double with modbus endianness: little endian ordered registers with big endian content.

Parameters:
value – float = 0.0: the initial value.

```
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() → float
```

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: float) → 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.

Rich Comparison

`oser.ByteStruct` and `oser.BitStruct` offer full support for rich comparisons.

Comparison is done by converting the members of both compared instances to lists and applying the compare operation on the lists. This leads to a hierarchical comparison in lexicographic order.

A comparison for `== None` always returns `False` and `!= None` always returns `True`.

Example

```
class Struct(oser.ByteStruct):
    def __init__(self):
        super(Struct, self).__init__()
        self.a = oser.UBInt8(1)
        self.b = oser.UBInt16(2)

a = Struct()
b = Struct()

a == b
# True

b.b.set(3)
a != b
# True
a < b
# True
a <= b
# True

a.a.set(23)
a > b
# True
a >= b
# True
```

Acceleration

In order to offer the best performance, the oser core consists of compiled machine code.

Oser types have pure Python wrappers to support development and allow IDEs to display function signatures and documentation.

To take full advantage of the optimized machine code at run-time, it is advisable to use the `oser.accelerator` module, which bypasses the Python wrappers.

The expected run-time acceleration is about `35 %`, by only changing a single line of code.

```
import oser  # Use for development; Contains Python wrappers
import oser.accelerator as oser  # Use for deployment; Does NOT contain Python wrappers

class MyDataStructure(oser.ByteStruct):
    def __init__(self):
        super(MyDataStructure, self).__init__()
        self.data = oser.UBInt8()
```

Fuzzing

Fuzzing or fuzz testing is a software testing technique that provides random, invalid or unexpected data to the inputs of a system. It is often automated or semi-automated. While providing data the system is checked to still work normally. Fuzzing proves the robustness of a system if it does not fail. If a test fails it may uncover bugs.

If a system has communication interfaces fuzzing can be used to test these interfaces and the software behind it.

Using OSER fuzzing testing is easy. The user only has to set fuzzing data to some fields of a struct that will be used and iterate over all combinations.

For each combination the data can be sent to the target and after that the target has to be checked to still work normally.

That's all.

Concepts

Fuzzing value combinations are generated by building the cartesian product of all fuzzing values.

For example you may have `a` and `b`. The fuzzing values for `a` are `[1, 2]` and the fuzzing values for `b` are `[3, 4]`.

The combinations are `[1, 3]`, `[1, 4]`, `[2, 3]` and `[2, 4]`. The number of combinations is the product of all value's lengths which is `4` in the example.

Values can be also random. Let the fuzzing values for `b` be `[random(), random()]` the cartesian product combinations are `[1, random()]`, `[1, random()]`, `[2, random()]` and `[2, random()]`.

In the example above always `4` combinations are used to be iterated for a fuzzing test.

With OSER it is possible to set fuzzing values using `oser.ByteType.set_fuzzing_values()` respectively `oser.BitType.set_fuzzing_values()` (which is inherited to all other types).

The fuzzing iterator is created by calling `oser.ByteStruct.fuzzing_iterator()` respectively `oser.BitStruct.fuzzing_iterator()`.

The first example from above looks like this:

```
>>> import oser

>>> class Struct(oser.ByteStruct):
...     def __init__(self):
...         super(Struct, self).__init__()
...         self.a = oser.UBInt8()
...         self.b = oser.UBInt8()
...
>>> struct = Struct()
>>> struct.a.set_fuzzing_values([1, 2])
>>> struct.b.set_fuzzing_values([3, 4])

>>> for product in struct.fuzzing_iterator():
...     print(product)
...
Struct():
    a: 1 (UBInt8)
    b: 3 (UBInt8)

Struct():
    a: 1 (UBInt8)
    b: 4 (UBInt8)

Struct():
    a: 2 (UBInt8)
    b: 3 (UBInt8)

Struct():
    a: 2 (UBInt8)
    b: 4 (UBInt8)
```

You can see that all cartesian product's combinations are created.

The second example with random values for `b` looks like:

```
>>> import oser

>>> class Struct(oser.ByteStruct):
...     def __init__(self):
...         super(Struct, self).__init__()
...         self.a = oser.UBInt8()
...         self.b = oser.UBInt8()
```

```
...
>>> struct = Struct()
>>> struct.a.set_fuzzing_values([1, 2])
>>> struct.b.set_fuzzing_values(
...     oser.RandomIntegerFuzzingValue(a=0, b=10, count=2))

>>> for product in struct.fuzzing_iterator():
...     print(product)
...
Struct():
    a: 1 (UBInt8)
    b: 5 (UBInt8)

Struct():
    a: 1 (UBInt8)
    b: 8 (UBInt8)

Struct():
    a: 2 (UBInt8)
    b: 4 (UBInt8)

Struct():
    a: 2 (UBInt8)
    b: 0 (UBInt8)
```

You can see that the value of `struct.b` is always random and that `4` combinations are generated, too.

The fuzzing values can also be a generator:

```
>>> import oser

>>> class Struct(oser.ByteStruct):
...     def __init__(self):
...         super(Struct, self).__init__()
...         self.a = oser.UBInt8()
...         self.b = oser.UBInt8()
...
>>> struct = Struct()
>>> struct.a.set_fuzzing_values([1, 2])


>>> def b_fuzzing_values():
...     yield 11
...     yield 12
...
>>> struct.b.set_fuzzing_values(b_fuzzing_values)

>>> for product in struct.fuzzing_iterator():
...     print(product)
...
Struct():
    a: 1 (UBInt8)
    b: 11 (UBInt8)

Struct():
    a: 1 (UBInt8)
```

```
    b: 12 (UBInt8)

Struct():
    a: 2 (UBInt8)
    b: 11 (UBInt8)

Struct():
    a: 2 (UBInt8)
    b: 12 (UBInt8)
```

Again `4` combinations are generated using the generated values of `b_fuzzing_values()` each twice.

Fuzzing also works on hierarchies:

```
>>> import oser

>>> class Struct(oser.ByteStruct):
...     def __init__(self):
...         super(Struct, self).__init__()
...         self.a = oser.UBInt8()
...         self.struct = oser.ByteStruct()
...         self.struct.a = oser.UBInt8()
...         self.struct.struct = oser.ByteStruct()
...         self.struct.struct.a = oser.UBInt8()
...
>>> struct = Struct()
>>> struct.a.set_fuzzing_values([1, 2])
>>> struct.struct.a.set_fuzzing_values([3, 4])
>>> struct.struct.struct.a.set_fuzzing_values([5, 6])

>>> for product in struct.fuzzing_iterator():
...     print(product)
...
Struct():
    a: 1 (UBInt8)
    struct: ByteStruct():
        a: 3 (UBInt8)
        struct: ByteStruct():
            a: 5 (UBInt8)

Struct():
    a: 1 (UBInt8)
    struct: ByteStruct():
        a: 3 (UBInt8)
        struct: ByteStruct():
            a: 6 (UBInt8)

Struct():
    a: 1 (UBInt8)
    struct: ByteStruct():
        a: 4 (UBInt8)
        struct: ByteStruct():
            a: 5 (UBInt8)

Struct():
    a: 1 (UBInt8)
    struct: ByteStruct():
```

```
        a: 4 (UBInt8)
        struct: ByteStruct():
            a: 6 (UBInt8)

Struct():
    a: 2 (UBInt8)
    struct: ByteStruct():
        a: 3 (UBInt8)
        struct: ByteStruct():
            a: 5 (UBInt8)

Struct():
    a: 2 (UBInt8)
    struct: ByteStruct():
        a: 3 (UBInt8)
        struct: ByteStruct():
            a: 6 (UBInt8)

Struct():
    a: 2 (UBInt8)
    struct: ByteStruct():
        a: 4 (UBInt8)
        struct: ByteStruct():
            a: 5 (UBInt8)

Struct():
    a: 2 (UBInt8)
    struct: ByteStruct():
        a: 4 (UBInt8)
        struct: ByteStruct():
            a: 6 (UBInt8)
```

Deep Copies

The fuzzing iterator supports deep copies of the iterated elements so that these can be stored for later use. If elements are not copied do not store them because every other iteration will overwrite the values.

Example:

```
>>> import oser

>>> class Struct(oser.ByteStruct):
...     def __init__(self):
...         super(Struct, self).__init__()
...         self.a = oser.UBInt8()
...
>>> struct = Struct()
>>> struct.a.set_fuzzing_values([1, 2])
>>> print(id(struct))
139796212522008
>>> for product in struct.fuzzing_iterator(copy=False):
...     print(id(product), product)  # the id is always the same
...
139796212522008 Struct():
    a: 1 (UBInt8)
```

```
139796212522008 Struct():
    a: 2 (UBInt8)

>>> for product in struct.fuzzing_iterator(copy=True):
...     print(id(product), product)  # the id is unique
...
139796212556800 Struct():
    a: 1 (UBInt8)

139796212568528 Struct():
    a: 2 (UBInt8)
```

If you set `copy=True` in `oser.ByteStruct.fuzzing_iterator()` the ids of `product` is not the id of `struct` (no aliases are created).

> **Warning**
>
> copying is slow since the element is encoded and decoded in background.

Fuzzing Random Data Adapters

Fuzzing Random Data Adapters help to build random fuzzing easily.

Random Integer Values

Using `oser.RandomIntegerFuzzingValue()` you can create random integer numbers.

```
>>> import oser

>>> class Struct(oser.ByteStruct):
...     def __init__(self):
...         super(Struct, self).__init__()
...         self.a = oser.UBInt8()
...         self.b = oser.UBInt8()
...
>>> struct = Struct()
>>> struct.a.set_fuzzing_values([1, 2])
>>> struct.b.set_fuzzing_values(
...     oser.RandomIntegerFuzzingValue(a=0, b=10, count=2))

>>> for product in struct.fuzzing_iterator():
...     print(product)
...
Struct():
    a: 1 (UBInt8)
    b: 5 (UBInt8)

Struct():
    a: 1 (UBInt8)
    b: 8 (UBInt8)

Struct():
    a: 2 (UBInt8)
    b: 4 (UBInt8)
```

```
Struct():
    a: 2 (UBInt8)
    b: 0 (UBInt8)
```

```
oser.RandomIntegerFuzzingValue(a: int, b: int, count: int, seed: int | None = None) →
Callable[[], Generator[int, None, None]]
```

A random integer fuzzing value generator.

Parameters:

- **a** (*int*) – the lower limit for `random.randint()`.
- **b** (*int*) – the upper limit for `random.randint()`.
- **count** (*int*) – the number of values to be generated.
- **seed=None** (*int*) – the seed for `random.seed()`. If `None` is supplied no seed is set.

Returns:
a callable that generates `count` random integer
values between `a` and `b`.

Return type:
callable

Random Float Values

Using `oser.RandomFloatFuzzingValue()` you can create random float values.

```
>>> import oser

>>> class Struct(oser.ByteStruct):
...     def __init__(self):
...         super(Struct, self).__init__()
...         self.a = oser.BFloat()
...
>>> struct = Struct()
>>> struct.a.set_fuzzing_values(
...     oser.RandomFloatFuzzingValue(factor=23.0, count=4)
... )

>>> for product in struct.fuzzing_iterator():
...     print(product)
...
Struct():
    a: 13.792093371685636 (BFloat)

Struct():
    a: 5.5145125854582 (BFloat)

Struct():
    a: 14.942608671817528 (BFloat)
```

```
Struct():
    a: 8.38521903857164 (BFloat)
```

```
oser.RandomFloatFuzzingValue(factor: float, count: int, seed: int | float | None = None) →
Callable[[], Generator[float, None, None]]
```

A random float fuzzing value generator. Random values are generated using `random.random()`.

Parameters:

- **factor** (*float*) – the factor the random value is multiplied by.
- **count** (*int*) – the number of values to be generated.
- **seed=None** (*int*) – the seed for `random.seed()`. If `None` is supplied no seed is set.

Returns:
a callable that generates `count` random float values

between `a` and `b`.

Return type:
callable

Random Bits

Using `oser.RandomBitsFuzzingValue()` you can create random bit values based on a mersenne twister.

```
>>> import oser

>>> class Struct(oser.BitStruct):
...     def __init__(self):
...         super(Struct, self).__init__()
...         self.data = oser.BitField(length=32)
...
>>> struct = Struct()
>>> struct.data.set_fuzzing_values(
...     oser.RandomBitsFuzzingValue(length=32, count=4)
... )

>>> for product in struct.fuzzing_iterator():
...     print(product)
...     encoded = product.encode()
...     print(oser.to_hex(encoded))
...
Struct():
    data: 3281267388 (BitField(32))

   0|  1|  2|  3
\xc3\x94\x2a\xbc
Struct():
    data: 2188825415 (BitField(32))

   0|  1|  2|  3
\x82\x76\xd3\x47
Struct():
```

```
    data: 2834423999 (BitField(32))

   0|  1|  2|  3
\xa8\xf1\xe0\xbf
Struct():
    data: 4167436224 (BitField(32))

   0|  1|  2|  3
\xf8\x66\x07\xc0
```

```
oser.RandomBitsFuzzingValue(length: int, count: int, seed: int | None = None) →
Callable[[], Generator[int, None, None]]
```

A random bits fuzzing value generator. Random values are generated using `random.getrandbits()`.

Parameters:

- **length** (*int*) – the number of bits to be generated.
- **count** (*int*) – the number of values to be generated.
- **seed=None** (*int*) – the seed for `random.seed()`. If `None` is supplied no seed is set.

Returns:
a callable that generates `count` random bits.
Return type:
callable

Random Strings

Using `oser.RandomStringFuzzingValue()` you can create random strings with printable letters.

```
>>> import oser
>>> import string


>>> class Struct(oser.ByteStruct):
...     def __init__(self):
...         super(Struct, self).__init__()
...         self.length = oser.UBInt8()
...         self.data = oser.String(length=lambda self: self.length.get())
...
>>> struct = Struct()
>>> struct.length.set_fuzzing_values(
...     oser.RandomIntegerFuzzingValue(a=0, b=10, count=2))
>>> struct.data.set_fuzzing_values(
...     oser.RandomStringFuzzingValue(length=10, count=2, chars=string.ascii_letters)
... )

>>> for product in struct.fuzzing_iterator():
...     print(product)
...
Struct():
    length: 3 (UBInt8)
    data: b'bZS'
```

```
Struct():
    length: 3 (UBInt8)
    data: b'KKg'

Struct():
    length: 8 (UBInt8)
    data: b'iUFEtnDF'

Struct():
    length: 8 (UBInt8)
    data: b'ksuQCkmg'
```

```
oser.RandomStringFuzzingValue(length: int, count: int, chars: str | None = None, seed: int | None = None) → Callable[[], Generator[bytes, None, None]]
```

A random string fuzzing value generator. Random values are generated using `random.choice()` with `chars`.

Parameters:

- **length** (*int*) – the number of characters to be generated..
- **count** (*int*) – the number of values to be generated.
- **chars=None** (*string*) – if set to `None` `string.printable` is used (digits, letters, punctuation and whispaces).
- **seed=None** (*int*) – the seed for `random.seed()`. If `None` is supplied no seed is set.

Returns:
a callable that generates `count` random string values.
Return type:
callable

Random Bytes

Using `oser.RandomStringFuzzingValue()` you can create random bytes.

```
>>> import oser

>>> class Struct(oser.ByteStruct):
...     def __init__(self):
...         super(Struct, self).__init__()
...         self.length = oser.UBInt8()
...         self.data = oser.String(length=lambda self: self.length.get())
...
>>> struct = Struct()
>>> struct.length.set_fuzzing_values(
...     oser.RandomIntegerFuzzingValue(a=0, b=10, count=2))
>>> struct.data.set_fuzzing_values(
...     oser.RandomBytesFuzzingValue(length=10, count=2)
... )

>>> for product in struct.fuzzing_iterator():
...     print(product)
...
```

```
Struct():
    length: 10 (UBInt8)
    data: b'\x15\xb0=\x14X\xa2\xf2}-\x02'

Struct():
    length: 10 (UBInt8)
    data: b'\x93)"\xd3\x13sM\xa6P\xa4'

Struct():
    length: 4 (UBInt8)
    data: b'ch0\xf5'

Struct():
    length: 4 (UBInt8)
    data: b'\xe2]\x88%'
```

```
oser.RandomBytesFuzzingValue(length: int, count: int, choices: List[int] | None = None,
seed: int | None = None) → Callable[[], Generator[bytes, None, None]]
```

A random bytes fuzzing value generator. Random values are generated using `random.choice()` with `choices`.

Parameters:

- **length** (*int*) – the number of bytes to be generated..
- **count** (*int*) – the number of values to be generated.
- **choices=None** (*list of int*) – if set to `None` `list(range(256))` is used.
- **seed=None** (*int*) – the seed for `random.seed()`. If `None` is supplied no seed is set.

Returns:
a callable that generates `count` random bytes values.
Return type:
callable

Special Elements

Special elements that check values while decoding, have sideeffects while encoding or are context-based elements are mentioned below.

The elements with sideeffects and checks are Checksum Types, `oser.Padding` and `oser.PaddingFlag`.

The special context-based elements are `oser.IfElse`, `oser.If` and `oser.Switch`.

CRC

CRCs can be used to add a checksum after fuzzed values or can be used with fuzzing values set, as well.

If no fuzzing values are set all CRCs work normally.

If fuzzing values are set the `strict` and the `automatic_calculation` features are temporarily disabled while the fuzzing iterator is alive.

```
>>> import oser

>>> class Struct(oser.ByteStruct):
...     def __init__(self):
...         super(Struct, self).__init__()
...         self.a = oser.UBInt16(23)
...         self.crc = oser.CRCB32(strict=True, automatic_calculation=True)
...
>>> struct = Struct()
>>> struct.a.set_fuzzing_values(
...     oser.RandomIntegerFuzzingValue(a=0, b=10, count=2)
... )

>>> # the crc works normally
... for product in struct.fuzzing_iterator():
...     encoded = product.encode()  # build crc
...     print(product)
...
Struct():
    a: 5 (UBInt16)
    crc: 1705890373 (CRCB32)

Struct():
    a: 8 (UBInt16)
    crc: 4142103048 (CRCB32)

>>> # the crc used as a fuzzing value
... struct.a.set_fuzzing_values(None)  # disable a fuzzing

>>> struct.crc.set_fuzzing_values(
...     oser.RandomIntegerFuzzingValue(a=0, b=1024, count=2)
... )

>>> for product in struct.fuzzing_iterator():
...     encoded = product.encode()  # build crc
...     print(product)
...
Struct():
    a: 8 (UBInt16)
    crc: 98 (CRCB32)

Struct():
    a: 8 (UBInt16)
    crc: 985 (CRCB32)

>>> # at the end the crc still works normally if no fuzzing values are set
... struct.crc.set_fuzzing_values(None)

>>> for product in struct.fuzzing_iterator():
...     encoded = product.encode()  # build crc
...     print(product)
...
Struct():
    a: 8 (UBInt16)
    crc: 4142103048 (CRCB32)
```

Padding

The `strict` feature of `oser.Padding` works normally while no fuzzing values are set and is temporarily disabled if fuzzing values are set.

```
>>> import oser

>>> class Struct(oser.ByteStruct):
...     def __init__(self):
...         super(Struct, self).__init__()
...         self.padding = oser.Padding(value=b"\x00", strict=True)
...
>>> struct = Struct()

>>> struct.padding.set_fuzzing_values(
...     oser.RandomBytesFuzzingValue(length=1, count=2)
... )

>>> for product in struct.fuzzing_iterator():
...     print(product)
...
Struct():
    padding: 206 (Padding)

Struct():
    padding: 90 (Padding)
```

PaddingFlag

The `strict` feature of `oser.PaddingFlag` works normally while no fuzzing values are set and is temporarily disabled if fuzzing values are set.

```
>>> import oser

>>> class Struct(oser.BitStruct):
...     def __init__(self):
...         super(Struct, self).__init__()
...         self.padding_flag = oser.PaddingFlag(value=False, strict=True)
...
>>> struct = Struct()

>>> struct.padding_flag.set_fuzzing_values(
...     oser.RandomBitsFuzzingValue(length=1, count=4)
... )

>>> for product in struct.fuzzing_iterator():
...     print(product)
...
Struct():
    padding_flag: 0 (PaddingFlag)

Struct():
    padding_flag: 1 (PaddingFlag)

Struct():
```

```
    padding_flag: 0 (PaddingFlag)

Struct():
    padding_flag: 1 (PaddingFlag)
```

IfElse and If

`oser.IfElse` and `oser.If` hide the values that depend on a condition.

To set fuzzing values these values can be accessed using `oser.IfElse.get_true_value()` (or `oser.If.get_true_value()`) respectively `oser.IfElse.get_false_value()` (or `oser.If.get_false_value()`).

```
>>> import oser

>>> class Struct(oser.ByteStruct):
...     def __init__(self):
...         super(Struct, self).__init__()
...         self.condition = oser.UBInt8(23)
...         self.data = oser.IfElse(
...             condition=lambda self: self.condition < 128,
...             if_true=oser.UBInt8(0),
...             if_false=oser.UBInt32(0xffffffff)
...         )
...
>>> struct = Struct()
>>> struct.condition.set_fuzzing_values(
...     oser.RandomIntegerFuzzingValue(a=0, b=255, count=4)
... )

>>> struct.data.get_true_value().set_fuzzing_values(
...     oser.RandomIntegerFuzzingValue(a=0, b=255, count=2)
... )

>>> struct.data.get_false_value().set_fuzzing_values(
...     oser.RandomIntegerFuzzingValue(a=0xfffffff0, b=0xffffffff, count=2)
... )

>>> for product in struct.fuzzing_iterator():
...     print(product)
...
Struct():
    condition: 124 (UBInt8)
    data: 42 (UBInt8)

Struct():
    condition: 124 (UBInt8)
    data: 208 (UBInt8)

Struct():
    condition: 63 (UBInt8)
    data: 34 (UBInt8)

Struct():
    condition: 63 (UBInt8)
    data: 223 (UBInt8)
```

```
Struct():
    condition: 5 (UBInt8)
    data: 84 (UBInt8)

Struct():
    condition: 5 (UBInt8)
    data: 251 (UBInt8)

Struct():
    condition: 192 (UBInt8)
    data: 4294967291 (UBInt32)

Struct():
    condition: 192 (UBInt8)
    data: 4294967295 (UBInt32)
```

Switch

`oser.Switch` hide the values that depend on a condition.

To set fuzzing values these values can be accessed using `oser.Switch.get_value()`.

```
>>> import oser

>>> class Message(oser.ByteStruct):
...     def __init__(self):
...         super(Message, self).__init__()
...         self.command = oser.UBInt8()
...         self.payload = oser.Switch(
...             condition=lambda self: self.command.get(),
...             values={
...                 0: oser.UBInt8(0),
...                 1: oser.UBInt16(1),
...                 2: oser.UBInt32(2),
...             },
...             default=oser.Nothing()
...             )
...

>>> message = Message()

>>> message.command.set_fuzzing_values(
...     oser.RandomIntegerFuzzingValue(a=0, b=2, count=5))

>>> # set fuzzing values for payload string
... message.payload.get_value(key=0).set_fuzzing_values(
...     oser.RandomIntegerFuzzingValue(a=0, b=10, count=2)
... )

>>> message.payload.get_value(key=1).set_fuzzing_values(
...     oser.RandomIntegerFuzzingValue(a=11, b=20, count=2)
... )

>>> message.payload.get_value(key=2).set_fuzzing_values(
...     oser.RandomIntegerFuzzingValue(a=21, b=30, count=2)
... )
```

```
>>> for product in message.fuzzing_iterator():
...     print(product)
...
Message():
    command: 0 (UBInt8)
    payload: 6 (UBInt8)

Message():
    command: 0 (UBInt8)
    payload: 4 (UBInt8)

Message():
    command: 2 (UBInt8)
    payload: 27 (UBInt32)

Message():
    command: 2 (UBInt8)
    payload: 24 (UBInt32)

Message():
    command: 2 (UBInt8)
    payload: 22 (UBInt32)

Message():
    command: 2 (UBInt8)
    payload: 30 (UBInt32)

Message():
    command: 1 (UBInt8)
    payload: 11 (UBInt16)

Message():
    command: 1 (UBInt8)
    payload: 20 (UBInt16)

Message():
    command: 2 (UBInt8)
    payload: 24 (UBInt32)

Message():
    command: 2 (UBInt8)
    payload: 27 (UBInt32)
```

Mixins

Fuzzing Type Mixin

The `oser.core.FuzzingTypeMixin` is used by every type in OSER. It lets you set the fuzzing values and offers fuzzing values iteration and setting the next fuzzing value.

```
class oser.core.FuzzingTypeMixin
```

A type mixin for fuzzing tests.

```
set_fuzzing_values(values: Generator[Any, None, None] | List[Any] | None) → None
```

Set fuzzing values.

Parameters:
values – the values used for fuzzing.

Fuzzing Struct Mixin

The `oser.core.FuzzingStructMixin` is used by `oser.ByteStruct` and `oser.BitStruct`. It offers a fuzzing iterator that is used to iterate all fuzzing combinations.

```
class oser.core.FuzzingStructMixin
```

A struct mixin for fuzzing tests.

```
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.

Examples

Random Integer Length for a String

`oser.RandomIntegerFuzzingValue()` can also be used in a lambda-expression for a `oser.String` length.

```
>>> import oser
>>> import string


>>> _random_length_iterator = oser.RandomIntegerFuzzingValue(a=0, b=10, count=1)


>>> class Struct(oser.ByteStruct):
...     def __init__(self):
...         super(Struct, self).__init__()
...         self.data = oser.String(length=lambda self: next(_random_length_iterator()))
...
```

```
>>> struct = Struct()
>>> struct.data.set_fuzzing_values(
...     oser.RandomStringFuzzingValue(length=10, count=4, chars=string.ascii_letters)
... )

>>> for product in struct.fuzzing_iterator():
...     print(product)
...
Struct():
    data: b's'

Struct():
    data: b''

Struct():
    data: b'CDJEsB'

Struct():
    data: b'WqOOOFldm'
```

Random Payload With a Valid CRC

```
>>> import oser
>>> import string


>>> class Struct(oser.ByteStruct):
...     def __init__(self):
...         super(Struct, self).__init__()
...         self.length = oser.UBInt8()
...         self.data = oser.String(length=lambda self: self.length.get())
...         self.crc = oser.CRCB32(strict=True, automatic_calculation=True)
...
>>> struct = Struct()
>>> struct.length.set_fuzzing_values(
...     oser.RandomIntegerFuzzingValue(a=0, b=10, count=2))
>>> struct.data.set_fuzzing_values(
...     oser.RandomStringFuzzingValue(length=10, count=2, chars=string.ascii_letters)
... )

>>> for product in struct.fuzzing_iterator():
...     encoded = product.encode()
...     print(product)
...     print(oser.to_hex(encoded))
...
Struct():
    length: 8 (UBInt8)
    data: b'SSosfNbZ'
    crc: 381554597 (CRCB32)

   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12
\x08\x53\x53\x6f\x73\x66\x4e\x62\x5a\x16\xbe\x0f\xa5
Struct():
    length: 8 (UBInt8)
    data: b'aKDKYIza'
    crc: 2493435128 (CRCB32)
```

```
   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11| 12
\x08\x61\x4b\x44\x4b\x59\x49\x7a\x61\x94\x9e\xcc\xf8
Struct():
    length: 3 (UBInt8)
    data: b'xyj'
    crc: 2464266468 (CRCB32)

   0|  1|  2|  3|  4|  5|  6|  7
\x03\x78\x79\x6a\x92\xe1\xb8\xe4
Struct():
    length: 3 (UBInt8)
    data: b'oyH'
    crc: 2247939372 (CRCB32)

   0|  1|  2|  3|  4|  5|  6|  7
\x03\x6f\x79\x48\x85\xfc\xd5\x2c
```

Inconsistent Messages

Lets assume you have a protocol to command a device. It consists of a field for a command, a complex command structure and a crc32 at the end. Every command needs a different payload length that is not encoded in the message.

The fuzzing test shall test valid and invalid command numbers with a random payload with an unexpected length. The crc at the end shall be valid.

To accomplish the complex `payload` is replaced by a simple `oser.String` with a random length (in each product). For the random values a `oser.RandomStringFuzzingValue()` is applied on the new `payload`.

```
>>> import oser
>>> import string


>>> def Payload():
...     payload = oser.Switch(
...         condition=lambda self: self.command.get(),
...         values={
...             0: oser.Nothing(),
...             1: oser.UBInt8(0),
...             2: oser.UBInt16(0),
...             3: oser.UBInt32(0),
...         },
...         default=oser.Nothing()
...         )
...     return payload
...

>>> class Message(oser.ByteStruct):
...     def __init__(self):
...         super(Message, self).__init__()
...         self.command = oser.UBInt8()
...         self.payload = Payload()
...         self.crc = oser.CRCB32(strict=True, automatic_calculation=True)
...
>>> message = Message()
```

```
>>> message.command.set_fuzzing_values(
...     oser.RandomIntegerFuzzingValue(a=0, b=10, count=4))
>>> # replace payload with a random length string
... _random_length_iterator = oser.RandomIntegerFuzzingValue(a=0, b=10, count=1)
>>> message.payload = oser.String(length=lambda self: next(_random_length_iterator()))

>>> # set fuzzing values for payload string
... message.payload.set_fuzzing_values(
...     oser.RandomStringFuzzingValue(length=10, count=2, chars=string.ascii_letters)
... )

>>> for product in message.fuzzing_iterator():
...     encoded = product.encode()  # to update the crc
...     print(product)
...
Message():
    command: 0 (UBInt8)
    payload: b'zmH'
    crc: 0 (CRCB32)

Message():
    command: 0 (UBInt8)
    payload: b'HGbgNbsSjC'
    crc: 1130147872 (CRCB32)

Message():
    command: 7 (UBInt8)
    payload: b'AKJBtT'
    crc: 237586712 (CRCB32)

Message():
    command: 7 (UBInt8)
    payload: b'x'
    crc: 2717259878 (CRCB32)

Message():
    command: 9 (UBInt8)
    payload: b'mJg'
    crc: 2778241945 (CRCB32)

Message():
    command: 9 (UBInt8)
    payload: b'RRjrE'
    crc: 1580841628 (CRCB32)

Message():
    command: 7 (UBInt8)
    payload: b'HEQExDgrmC'
    crc: 1477774535 (CRCB32)

Message():
    command: 7 (UBInt8)
    payload: b'XnLI'
    crc: 2554068935 (CRCB32)
```

Full Fuzzing Example

In this example a `send` and a `check` method is added to demonstrate how a real fuzzing test can look like.

The same protocol already mentioned is reused here.

```
>>> import oser
>>> import string


>>> def Payload():
...     payload = oser.Switch(
...         condition=lambda self: self.command.get(),
...         values={
...             0: oser.Nothing(),
...             1: oser.UBInt8(0),
...             2: oser.UBInt16(0),
...             3: oser.UBInt32(0),
...         },
...         default=oser.Nothing()
...         )
...     return payload
...

>>> class Message(oser.ByteStruct):
...     def __init__(self):
...         super(Message, self).__init__()
...         self.command = oser.UBInt8()
...         self.payload = Payload()
...         self.crc = oser.CRCB32(strict=True, automatic_calculation=True)
...
>>> message = Message()
>>> message.command.set_fuzzing_values(
...     oser.RandomIntegerFuzzingValue(a=0, b=10, count=2000))
>>> # replace payload with a random length string
... _random_length_iterator = oser.RandomIntegerFuzzingValue(a=0, b=10, count=1)
>>> message.payload = oser.String(length=lambda self: next(_random_length_iterator()))

>>> # set fuzzing values for payload string
... message.payload.set_fuzzing_values(
...     oser.RandomStringFuzzingValue(length=10, count=2000, chars=string.ascii_letters)
... )


>>> def send(binary_data):
...     """
...     This method send the ``binary_data`` to
...     the target.
...     """
...     # ...
...

>>> def check():
...     """
...     This method checks if the target still works.
...
...     Raises:
```

```
...         Exception: if the target does not work properly anymore
...     """
...
>>> for product in message.fuzzing_iterator():
...     encoded = product.encode()  # to update the crc
...     print(product)
...     print(oser.to_hex(encoded))
...     send(encoded)
...     check()
...
Message():
    command: 9 (UBInt8)
    payload: b'tsLpqQCiWR'
    crc: 3896448329 (CRCB32)

   0|  1|  2|  3|  4
\x09\xe8\x3f\x15\x49
Message():
    command: 9 (UBInt8)
    payload: b'zvVSZoCpSC'
    crc: 488979088 (CRCB32)

   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11
\x09\x7a\x76\x56\x53\x5a\x6f\x43\x1d\x25\x3a\x90
Message():
    command: 1 (UBInt8)
    payload: b'gAzRLVBRka'
    crc: 4032940286 (CRCB32)

   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11
\x01\x67\x41\x7a\x52\x4c\x56\x42\xf0\x61\xc8\xfe
Message():
    command: 1 (UBInt8)
    payload: b'vYNceTLpIY'
    crc: 795191758 (CRCB32)

   0|  1|  2|  3|  4|  5|  6|  7
\x01\x76\x59\x4e\x2f\x65\xa9\xce
Message():
    command: 0 (UBInt8)
    payload: b'LWHwQuNRtl'
    crc: 3587833413 (CRCB32)

   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11
\x00\x4c\x57\x48\x77\x51\x75\x4e\xd5\xd9\xfe\x45
Message():
    command: 0 (UBInt8)
    payload: b'DvzLhAaxbz'
    crc: 2470879835 (CRCB32)

   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11
\x00\x44\x76\x7a\x4c\x68\x41\x61\x93\x46\xa2\x5b
Message():
    command: 1 (UBInt8)
    payload: b'AQFkyHKKwp'
    crc: 680941664 (CRCB32)

   0|  1|  2|  3|  4|  5|  6|  7|  8|  9| 10| 11
\x01\x41\x51\x46\x6b\x79\x48\x4b\x28\x96\x58\x60
Message():
```

```
    command: 1 (UBInt8)
    payload: b'uUhByovAZf'
    crc: 165847963 (CRCB32)

   0|  1|  2|  3|  4|  5|  6|  7
\x01\x75\x55\x68\x09\xe2\xa3\x9b

# and so on
```

Utilities

```
oser.util.to_hex(data: bytes) → str
```

Return a string converted to hex with indexes above.

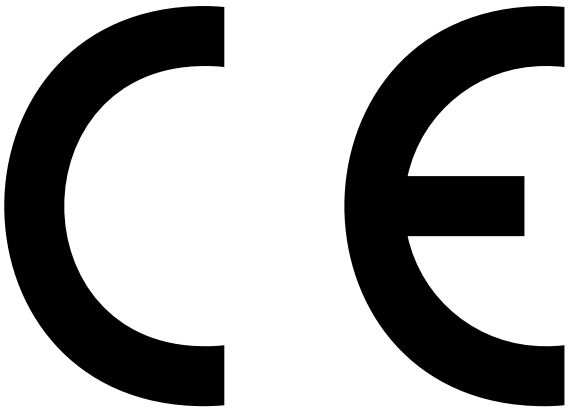
Parameters:
data (*str*) – the data to be encoded to a hexadecimal format.

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

oser-3.3.0:

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

oser-3.2.0:

- implement Cyber Resilience Act Conformity (CRA)
- update dependencies
- update to `protobuf==7.35.1`

oser-3.1.6:

- add support for Python 3.14
- update to `protobuf~=6.32.1`

oser-3.1.5:

- add support for Python 3.13

oser-3.1.4:

- add build environment data for validation purposes

oser-3.1.3:

- add support for Python 3.12

oser-3.1.2:

- update to `protobuf~=4.23.2`

oser-3.1.1:

- fix typing error

oser-3.1.0:

- add type-hints
- use `hlm-3.1`
- update to `protobuf~=4.22.1`

oser-3.0.1:

- add support for Python 3.11

oser-3.0.0:

> **Warning**
>
> oser-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.

- update to `hlm-3.0.0`
- rename `size()` and `byte_size()` to `get_size()` and `get_byte_size()`

oser-2.x

- oser-2 was never released. Instead we decided to release oser-3 which only support Python 3.x so the version fits better.

oser-1.3.10

> **Warning**
> oser-1.3 reached the end of life on January 1st, 2022. Please upgrade to oser-3.0 which can be used with little changes.

- use `hlm-2.0.1`

oser-1.3.9

- reduce wheel size and improve performance
- CRC fuzzing disables strict mode and automatic calculation while fuzzing
- update to `protobuf~=3.17.3`

oser-1.3.8

- remove docs from wheels

oser-1.3.7

- update to `hlm-2.0`

oser-1.3.6

- modernize documentation
- fix examples that were not working with Python 3.x

oser-1.3.5

- add support for Python 3.9

oser-1.3.4:

- build manylinux2014 wheels for Python 3.x and manylinux2010 wheels for Python 2.7

oser-1.3.3:

- > **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 (oser 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 `protobuf~=3.11.2`

oser-1.3.2:

- fix pickleing on Python 2.7
- fix GeneratorExit in `oser.Switch` and `oser.IfElse`
- fix `oser.Array.to_dict()` and `oser.RepeatUntil.to_dict()`

oser-1.3.1:

- update to hlm-1.0.4

oser-1.3.0:

- add support for Rich Comparison
- add types for modbus (see Modbus Types)
- add `format` parameter to all integer types and CRCs
- default format for CRCs is `"hex"`
- add `oser.BitMap`
- `oser.ByteStruct` and `oser.BitStruct` can be converted to `dict` using `oser.ByteStruct.to_dict()` and `oser.BitStruct.to_dict()`
- `oser.ByteStruct` and `oser.BitStruct` can be loaded from `dict` using `oser.ByteStruct.from_dict()` and `oser.BitStruct.from_dict()`
- `root()` now calls `up()` and thus traversing a hierarchy can be stopped if needed by overwriting `up()` somewhere returning `None`
- a `KeyError` in `oser.Switch()` now leads to a `oser.DecodeException`
- `oser.Flag` and `oser.PaddingFlag` cast all values to `bool`
- add support for bytes and strings for `oser.Padding`, `oser.String`, `oser.Data` and `oser.RegularExpressionMatch`
- improve output of introspection
- improve some docstrings
- update to `hlm-1.0.2`

oser-1.2.9:

> **Warning**
>
> oser-1.2 reached the end of its life on January 1st, 2021. Please upgrade to oser-1.3 which is fully backwards compatible to oser-1.2.

- build manylinux2014 wheels for Python 3.x and manylinux2010 wheels for Python 2.7

oser-1.2.8:

- > **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 (oser 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 `protobuf~=3.11.2`

oser-1.2.7:

- fix pickleing on Python 2.7

oser-1.2.6:

- update to hlm-0.21.0

oser-1.2.5:

- KeyError in `oser.Switch` leads to a readable DecodeException

oser-1.2.4:

- fix deletion of members in structs

oser-1.2.3:

- fix arithmetics with struct members

oser-1.2.2:

- update to hlm-0.18.0

oser-1.2.1:

- fix `copy.deepcopy(object)` to work on unencodable objects

oser-1.2.0:

- add support for Python 3.7
- improve bit-type encoding and decoding performance by orders of magnitude
- using Acceleration allows to easily improve encoding and decoding performance

- oser nodes can be pickled (used with `pickle`)
- add `EncodeException` and `DecodeException` to make exceptions more obvious
- `oser.String` and `oser.Data` check parameters and raise exceptions
- improve oser version output
- improve internal build process with Docker
- update to hlm-0.17.0

oser-1.1.13:

> **Warning**
> oser-1.1 reached the end of its life on January 1st, 2021. Please upgrade to oser-1.3 which is fully backwards compatible to oser-1.1.

- build manylinux2014 wheels for Python 3.x and manylinux2010 wheels for Python 2.7

oser-1.1.12:

- > **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 (oser 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 `protobuf~=3.11.2`

oser-1.1.11:

- update to hlm-0.21.0

oser-1.1.10:

- build wheels for Python 3.7

oser-1.1.9:

- fix arithmetics with struct members

oser-1.1.8:

- update to hlm-0.18.0

oser-1.1.7:

- fix introspect with `oser.Value`
- better error messages during decoding for `oser.Constant`, `oser.Padding`, `oser.Enum`, `oser.Array`, `oser.ByteStruct` and `oser.BitStruct`
- `oser.Constant` raises `TypeError` when called with insufficient arguments

oser-1.1.6:

- update to hlm-0.15.0

oser-1.1.5:

- add missing wrapper for the Fuzzing feature
- update to hlm-0.12.0 to fix the bug that some usb-sticks changed the host id
- update to protobuf-3.6.0
- wheels for Linux are delivered as manylinux-wheels and can be run on most linux distributions
- add `hst` (HILSTER Support Tool) as a dependency

oser-1.1.4:

- fixed context data generation

oser-1.1.3:

- fixed comparison of oser instances for equality
- improve exceptions in conditional elements

oser-1.1.2:

- fixed a bug in `oser.Switch` when using `oser.Nothing` as the default value
- AttributeErrors in conditional elements are more obvious

oser-1.1.1:

- fixed bugs concerning size calculation when using conditional elements and bit-types

oser-1.1.0:

- added support for Python 3.6
- added the Fuzzing feature, documentation and examples.

- added `oser.RandomIntegerFuzzingValue()` fuzzing adapter.
- added `oser.RandomFloatFuzzingValue()` fuzzing adapter.
- added `oser.RandomBitsFuzzingValue()` fuzzing adapter.
- added `oser.RandomStringFuzzingValue()` fuzzing adapter.
- added `oser.RandomBytesFuzzingValue()` fuzzing adapter.
- added `oser.core.FuzzingTypeMixin`.
- added `oser.core.FuzzingStructMixin`.
- added `oser.IfElse.set_true_value()` to set the value for a `True` `condition`.
- added `oser.IfElse.set_false_value()` to set the value for a `False` `condition`.
- added `oser.IfElse.get_true_value()` to get the value for a `True` `condition`.
- added `oser.IfElse.get_false_value()` to get the value for a `False` `condition`.
- fixed `oser.IfElse._items()`.
- added `oser.Switch.set_value()` to access conditional values.
- added `oser.Switch.get_value()` to access conditional values.
- added `oser.Switch.get_items()` to iterate conditional keys and values.
- added `oser.Switch.get_keys()` to iterate conditional keys.
- added `oser.Switch.get_values()` to iterate conditional values.
- added `oser.Switch.delete_value()` to delete a value.
- added `oser.Switch.update_values()` to update the values.
- fixed `oser.Switch._items()`.
- `oser.Padding.get()` returns the value instead of a string and an interger value can be used for padding.
- `copy.deepcopy()` can be used to create deep copies of oser instances.
- add methods to enable and disable the checksum and crc automatic calculation feature, e.g. `oser.CRCL8.set_automatic_calculation()`.
- `oser.Array` automatically resizes when a value is set.
- `oser.Array` supports slicing when multiple values are set.
- `oser.RepeatUntil` automatically resizes when a value is set.
- `oser.RepeatUntil` supports slicing when multiple values are set.

oser-1.0.6:

- fixed context data generation

oser-1.0.5:

- fixed comparison of oser instances for equality
- improve exceptions in conditional elements

oser-1.0.4:

- fixed a bug in `oser.Switch` when using `oser.Nothing` as the default value

oser-1.0.3:

- fixed bugs concerning size calculation when using conditional elements and bit-types

oser-1.0.2:

- fixed a bug with conditional fields within an `oser.BitStruct`
- `oser.Padding.get()` returns the value and not a string
- fixed a bug in `oser.Switch._items()`, `oser.IfElse._items()` and `oser.If._items()`
- `oser.Nothing` returns `None` instead of an empty string

oser-1.0.1:

- `oser.ByteStruct.size()` calculation was wrong for embedded `oser.BitStruct` members.
- when overwriting internal methods with fields an exception is raised
- fixed the context data in `oser.Array` and `oser.RepeatUntil`
- added `oser.Switch._items()`, `oser.IfElse._items()`, `oser.If._items()`, `oser.Array._items()` and `oser.RepeatUntil._items()` that generates name-item tuples
- added `oser.Enum.get_value()` to get the current raw value
- fixed some docstrings
- updated to `protobuf~=3.1.0`
- updated to `hlm~=0.10.3`

oser-1.0.0:

- added `oser.If`
- added `oser.Nothing`
- added `oser.RepeatUntil`
- added `oser.Data`
- added Python 3.5 support
- support for Python 2.7 and Python 3.5 using the same code base
- docs are included in wheel (run `oser docs` from commandline)
- updated hlm to use the new license version (enabling features, banners, etc.)
- simplified import strings
- improved continuous integration for oser

- implemented continuous delivery for oser updates
- added pure python wrappers where needed
- better pep8 compatibility
- `oser.Switch.get()` and `oser.IfElse.get()` return the result of `.get()` applied on the currently selected object
- `oser.Switch.get_current()` and `oser.IfElse.get_current()` can be used to retrieve the currently selected object
- added `oser.__version_info__` (tuple) to compare the version against a required one
- improved docs
- using hilster_theme-0.9 in docs

Indices and tables

- Index
- Module Index
- Search Page