
bitstruct Documentation

Release 8.17.0

Erik Moqvist

Feb 17, 2023

Contents

1	About	1
2	Installation	3
3	Performance	5
4	MicroPython	7
5	Example usage	9
6	Contributing	13
7	Functions	15
8	Classes	19
	Index	21

CHAPTER 1

About

This module is intended to have a similar interface as the python struct module, but working on bits instead of primitive data types (char, int, ...).

Project homepage: <https://github.com/erimoq/bitstruct>

Documentation: <https://bitstruct.readthedocs.io>

CHAPTER 2

Installation

```
pip install bitstruct
```


Parts of this package has been re-implemented in C for faster pack and unpack operations. There are two independent C implementations; *bitstruct.c*, which is part of this package, and the standalone package `cbitstruct`. These implementations are only available in CPython 3, and must be explicitly imported. By default the pure Python implementation is used.

To use *bitstruct.c*, do `import bitstruct.c` as `bitstruct`.

To use `cbitstruct`, do `import cbitstruct` as `bitstruct`.

bitstruct.c has a few limitations compared to the pure Python implementation:

- Integers and booleans must be 64 bits or less.
- Text and raw must be a multiple of 8 bits.
- Bit endianness and byte order are not yet supported.
- `byteswap()` can only swap 1, 2, 4 and 8 bytes.

See `cbitstruct` for its limitations.

CHAPTER 4

MicroPython

The C implementation has been ported to [MicroPython](#). See [bitstruct-micropython](#) for more details.

Example usage

A basic example of packing and unpacking four integers using the format string 'ulu3u4s16':

```
>>> from bitstruct import *
>>> pack('ulu3u4s16', 1, 2, 3, -4)
b'\xa3\xff\xfc'
>>> unpack('ulu3u4s16', b'\xa3\xff\xfc')
(1, 2, 3, -4)
>>> calcsize('ulu3u4s16')
24
```

An example compiling the format string once, and use it to pack and unpack data:

```
>>> import bitstruct
>>> cf = bitstruct.compile('ulu3u4s16')
>>> cf.pack(1, 2, 3, -4)
b'\xa3\xff\xfc'
>>> cf.unpack(b'\xa3\xff\xfc')
(1, 2, 3, -4)
```

Use the `pack_into` and `unpack_from` functions to pack/unpack values at a bit offset into the data, in this example the bit offset is 5:

```
>>> from bitstruct import *
>>> data = bytearray(b'\x00\x00\x00\x00')
>>> pack_into('ulu3u4s16', data, 5, 1, 2, 3, -4)
>>> data
bytearray(b'\x05\x1f\xff\xe0')
>>> unpack_from('ulu3u4s16', data, 5)
(1, 2, 3, -4)
```

The unpacked values can be named by assigning them to variables or by wrapping the result in a named tuple:

```
>>> from bitstruct import *
>>> from collections import namedtuple
```

(continues on next page)

(continued from previous page)

```

>>> MyName = namedtuple('myname', ['a', 'b', 'c', 'd'])
>>> unpacked = unpack('ulu3u4s16', b'\xa3\xff\xfc')
>>> myname = MyName(*unpacked)
>>> myname
myname(a=1, b=2, c=3, d=-4)
>>> myname.c
3

```

Use the `pack_dict` and `unpack_dict` functions to pack/unpack values in dictionaries:

```

>>> from bitstruct import *
>>> names = ['a', 'b', 'c', 'd']
>>> pack_dict('ulu3u4s16', names, {'a': 1, 'b': 2, 'c': 3, 'd': -4})
b'\xa3\xff\xfc'
>>> unpack_dict('ulu3u4s16', names, b'\xa3\xff\xfc')
{'a': 1, 'b': 2, 'c': 3, 'd': -4}

```

An example of `packing` and `unpacking` an unsigned integer, a signed integer, a float, a boolean, a byte string and a string:

```

>>> from bitstruct import *
>>> pack('u5s5f32b1r13t40', 1, -1, 3.75, True, b'\xff\xff', 'hello')
b'\x0f\xd0\x1c\x00\x00?\xffhello'
>>> unpack('u5s5f32b1r13t40', b'\x0f\xd0\x1c\x00\x00?\xffhello')
(1, -1, 3.75, True, b'\xff\xff', 'hello')
>>> calcsize('u5s5f32b1r13t40')
96

```

The same format string and values as in the previous example, but using LSB (Least Significant Bit) first instead of the default MSB (Most Significant Bit) first:

```

>>> from bitstruct import *
>>> pack('<u5s5f32b1r13t40', 1, -1, 3.75, True, b'\xff\xff', 'hello')
b'\x87\xc0\x00\x03\x80\xbf\xff\xf666\xa6\x16'
>>> unpack('<u5s5f32b1r13t40', b'\x87\xc0\x00\x03\x80\xbf\xff\xf666\xa6\x16')
(1, -1, 3.75, True, b'\xff\xff', 'hello')
>>> calcsize('<u5s5f32b1r13t40')
96

```

An example of `unpacking` values from a hexstring and a binary file:

```

>>> from bitstruct import *
>>> from binascii import unhexlify
>>> unpack('s17s13r24', unhexlify('0123456789abcdef'))
(582, -3751, b'\xe2j\xf3')
>>> with open("test.bin", "rb") as fin:
...     unpack('s17s13r24', fin.read(8))
...
...
(582, -3751, b'\xe2j\xf3')

```

Change endianness of the data with `byteswap`, and then `unpack` the values:

```

>>> from bitstruct import *
>>> packed = pack('ulu3u4s16', 1, 2, 3, 1)
>>> unpack('ulu3u4s16', byteswap('12', packed))
(1, 2, 3, 256)

```

A basic example of `packing` and `unpacking` four integers using the format string `'ulu3u4s16'` using the C implementation:

```
>>> from bitstruct.c import *
>>> pack('ulu3u4s16', 1, 2, 3, -4)
b'\xa3\xff\xfc'
>>> unpack('ulu3u4s16', b'\xa3\xff\xfc')
(1, 2, 3, -4)
```


CHAPTER 6

Contributing

1. Fork the repository.
2. Install prerequisites.

```
pip install -r requirements.txt
```

3. Implement the new feature or bug fix.
4. Implement test case(s) to ensure that future changes do not break legacy.
5. Run the tests.

```
make test
```

6. Create a pull request.

`bitstruct.pack` (*fmt*, **args*)

Return a bytes object containing the values *v1*, *v2*, ... packed according to given format string *fmt*. If the total number of bits are not a multiple of 8, padding will be added at the end of the last byte.

fmt is a string of bit order-type-length groups, and optionally a byte order identifier after the groups. Bit Order and byte order may be omitted.

Bit Order is either > or <, where > means MSB first and < means LSB first. If bit order is omitted, the previous values' bit order is used for the current value. For example, in the format string 'u1<u2u3', u1 is MSB first and both u2 and u3 are LSB first.

Byte Order is either > or <, where > means most significant byte first and < means least significant byte first. If byte order is omitted, most significant byte first is used.

There are eight types; u, s, f, b, t, r, p and P.

- u – unsigned integer
- s – signed integer
- f – floating point number of 16, 32, or 64 bits
- b – boolean
- t – text (ascii or utf-8)
- r – raw, bytes
- p – padding with zeros, ignore
- P – padding with ones, ignore

Length is the number of bits to pack the value into.

Example format string with default bit and byte ordering: 'u1u3p7s16'

Same format string, but with least significant byte first: 'u1u3p7s16<'

Same format string, but with LSB first (< prefix) and least significant byte first (< suffix): '<u1u3p7s16<'

It is allowed to separate groups with a single space for better readability.

`bitstruct.unpack` (*fmt*, *data*, *allow_truncated=False*, *text_encoding='utf-8'*, *text_errors='strict'*)
Unpack *data* (bytes or bytearray) according to given format string *fmt*.

If *allow_truncated* is *True*, *data* may be shorter than the number of items specified by *fmt*; in this case, only the complete items will be unpacked. The result is a tuple even if it contains exactly one item.

Text fields are decoded with given encoding *text_encoding* and error handling as given by *text_errors* (both passed to `bytes.decode()`).

`bitstruct.pack_into` (*fmt*, *buf*, *offset*, **args*, ***kwargs*)

Pack given values *v1*, *v2*, ... into given bytearray *buf*, starting at given bit offset *offset*. Pack according to given format string *fmt*. Give *fill_padding* as *False* to leave padding bits in *buf* unmodified.

`bitstruct.unpack_from` (*fmt*, *data*, *offset=0*, *allow_truncated=False*, *text_encoding='utf-8'*, *text_errors='strict'*)

Unpack *data* (bytes or bytearray) according to given format string *fmt*, starting at given bit offset *offset*. If *allow_truncated* is *True*, *data* may be shorter than the number of items specified by *fmt*; in this case, only the complete items will be unpacked. The result is a tuple even if it contains exactly one item.

`bitstruct.pack_dict` (*fmt*, *names*, *data*)

Same as `pack()`, but data is read from a dictionary.

The names list *names* contains the format group names, used as keys in the dictionary.

```
>>> pack_dict('u4u4', ['foo', 'bar'], {'foo': 1, 'bar': 2})
b'\x12'
```

`bitstruct.unpack_dict` (*fmt*, *names*, *data*, *allow_truncated=False*, *text_encoding='utf-8'*, *text_errors='strict'*)

Same as `unpack()`, but returns a dictionary.

See `pack_dict()` for details on *names*.

```
>>> unpack_dict('u4u4', ['foo', 'bar'], b'\x12')
{'foo': 1, 'bar': 2}
```

`bitstruct.pack_into_dict` (*fmt*, *names*, *buf*, *offset*, *data*, ***kwargs*)

Same as `pack_into()`, but data is read from a dictionary.

See `pack_dict()` for details on *names*.

`bitstruct.unpack_from_dict` (*fmt*, *names*, *data*, *offset=0*, *allow_truncated=False*, *text_encoding='utf-8'*, *text_errors='strict'*)

Same as `unpack_from()`, but returns a dictionary.

See `pack_dict()` for details on *names*.

`bitstruct.calcsize` (*fmt*)

Return the number of bits in given format string *fmt*.

```
>>> calcsize('uls3p4')
8
```

`bitstruct.byteswap` (*fmt*, *data*, *offset=0*)

Swap bytes in *data* according to *fmt*, starting at byte *offset* and return the result. *fmt* must be an iterable, iterating over number of bytes to swap. For example, the format string '24' applied to the bytes `b'\x00\x11\x22\x33\x44\x55'` will produce the result `b'\x11\x00\x55\x44\x33\x22'`.

`bitstruct.compile` (*fmt*, *names=None*, *text_encoding='utf-8'*, *text_errors='strict'*)

Compile given format string *fmt* and return a compiled format object that can be used to pack and/or unpack data multiple times.

Returns a *CompiledFormat* object if *names* is `None`, and otherwise a *CompiledFormatDict* object.

See *pack_dict()* for details on *names*.

See *unpack()* for details on *text_encoding* and *text_errors*.

class `bitstruct.CompiledFormat` (*fmt*, *text_encoding='utf-8'*, *text_errors='strict'*)

A compiled format string that can be used to pack and/or unpack data multiple times.

Instances of this class are created by the factory function `compile()`.

pack (**args*)

See `pack()`.

unpack (*data*, *allow_truncated=False*)

See `unpack()`.

pack_into (*buf*, *offset*, **args*, ***kwargs*)

See `pack_into()`.

unpack_from (*data*, *offset=0*, *allow_truncated=False*)

See `unpack_from()`.

class `bitstruct.CompiledFormatDict` (*fmt*, *names=None*, *text_encoding='utf-8'*,
text_errors='strict')

See `CompiledFormat`.

pack (*data*)

See `pack_dict()`.

unpack (*data*, *allow_truncated=False*)

See `unpack_dict()`.

pack_into (*buf*, *offset*, *data*, ***kwargs*)

See `pack_into_dict()`.

unpack_from (*data*, *offset=0*, *allow_truncated=False*)

See `unpack_from_dict()`.

B

`byteswap()` (in module *bitstruct*), 16

C

`calcsize()` (in module *bitstruct*), 16

`compile()` (in module *bitstruct*), 16

`CompiledFormat` (class in *bitstruct*), 19

`CompiledFormatDict` (class in *bitstruct*), 19

P

`pack()` (*bitstruct.CompiledFormat* method), 19

`pack()` (*bitstruct.CompiledFormatDict* method), 19

`pack()` (in module *bitstruct*), 15

`pack_dict()` (in module *bitstruct*), 16

`pack_into()` (*bitstruct.CompiledFormat* method), 19

`pack_into()` (*bitstruct.CompiledFormatDict* method),
19

`pack_into()` (in module *bitstruct*), 16

`pack_into_dict()` (in module *bitstruct*), 16

U

`unpack()` (*bitstruct.CompiledFormat* method), 19

`unpack()` (*bitstruct.CompiledFormatDict* method), 19

`unpack()` (in module *bitstruct*), 16

`unpack_dict()` (in module *bitstruct*), 16

`unpack_from()` (*bitstruct.CompiledFormat* method),
19

`unpack_from()` (*bitstruct.CompiledFormatDict*
method), 19

`unpack_from()` (in module *bitstruct*), 16

`unpack_from_dict()` (in module *bitstruct*), 16