pcstools Documentation

Release 0.4

2019, Mathias Svensson

May 07, 2019

Contents

The primary location for this documentation is at pcstools.readthedocs.io.

See our https://github.com/DIKU-PCS/pcstools for more information.

CHAPTER 1

Module Index

Each of the `pcstools` modules is documented here.

1.1 `pcstools.packing.little_endian` — Packing and unpacking of little-endian integers

`pcstools.packing.little_endian.`**`pack8`**(*arg*)

Packs a thing into a bytestring, while recursively decending into iterables.

Integers are packed as 8-bit, unsigned, little-endian values.

Byte-strings and byte-arrays are packed as-is.

Unicode-strings are utf-8 encoded.

Examples

```
>>> from pcstools.packing.little_endian import pack8
>>> assert pack8(0x41) == b'A'
>>> assert pack8([0x41, 0x42]) == b'AB'
>>> assert pack8([0x41, 0x42, (u'hello', b'hi', bytearray(b'sup'))]) == b
↪'ABhellohisup'
>>> assert pack8(u'') == b'\xe2\x98\x83'
>>> pack8(-1)
Traceback (most recent call last):
   ...
ValueError: Number must be positive and below 2**8, but number == -1
>>> # It is also possible to combine pack8 with other packers
>>> from pcstools.packing.little_endian import pack32
>>> assert pack8([0x41, 0x42, pack32(0xdeadbeef)]) == b'AB\xef\xbe\xad\xde'
```

pcstools.packing.little_endian.**pack16**(*arg*)

Packs a thing into a bytestring, while recursively decending into iterables.

Integers are packed as 16-bit, unsigned, little-endian values.

Byte-strings and byte-arrays are packed as-is.

Unicode-strings are utf-8 encoded.

Examples

```
>>> from pcstools.packing.little_endian import pack16
>>> assert pack16(0x4241) == b'AB'
>>> assert pack16([0x4241, 0x4142]) == b'ABBA'
>>> assert pack16([0x4241, (u'hello', b'hi', bytearray(b'sup'))]) == b
↪'ABhellohisup'
>>> assert pack16(u'') == b'\xe2\x98\x83'
>>> pack16(-1)
Traceback (most recent call last):
  ...
ValueError: Number must be positive and below 2**16, but number == -1
>>> # It is also possible to combine pack16 with other packers
>>> from pcstools.packing.little_endian import pack32
>>> assert pack16([0x4241, pack32(0xdeadbeef)]) == b'AB\xef\xbe\xad\xde'
```

pcstools.packing.little_endian.**pack32**(*arg*)

Packs a thing into a bytestring, while recursively decending into iterables.

Integers are packed as 32-bit, unsigned, little-endian values.

Byte-strings and byte-arrays are packed as-is.

Unicode-strings are utf-8 encoded.

Examples

```
>>> from pcstools.packing.little_endian import pack32
>>> assert pack32(0x44434241) == b'ABCD'
>>> assert pack32([0x44434241, 0x41424344]) == b'ABCDDCBA'
>>> assert pack32([0x44434241, (u'hello', b'hi', bytearray(b'sup'))]) == b
↪'ABCDhellohisup'
>>> assert pack32(u'') == b'\xe2\x98\x83'
>>> pack32(-1)
Traceback (most recent call last):
  ...
ValueError: Number must be positive and below 2**32, but number == -1
>>> # It is also possible to combine pack32 with other packers
>>> from pcstools.packing.little_endian import pack8
>>> assert pack32([0xdeadbeef, pack8(0x41)]) == b'\xef\xbe\xad\xdeA'
```

pcstools.packing.little_endian.**pack64**(*arg*)

Packs a thing into a bytestring, while recursively decending into iterables.

Integers are packed as 64-bit, unsigned, little-endian values.

Byte-strings and byte-arrays are packed as-is.

Unicode-strings are utf-8 encoded.

Examples

```
>>> from pcstools.packing.little_endian import pack64
>>> assert pack64(0x4847464544434241) == b'ABCDEFGH'
>>> assert pack64([0x4847464544434241, 0x4142434445464748]) == b'ABCDEFGHHGFEDCBA'
>>> assert pack64([0x4847464544434241, (u'hello', b'hi', bytearray(b'sup'))]) == b
↪'ABCDEFGHhellohisup'
>>> assert pack64(u'') == b'\xe2\x98\x83'
>>> pack64(-1)
Traceback (most recent call last):
   ...
ValueError: Number must be positive and below 2**64, but number == -1
>>> # It is also possible to combine pack64 with other packers
>>> from pcstools.packing.little_endian import pack8
>>> assert pack64([0xdeadbeefdeadbeef, pack8(0x41)]) == b
↪'\xef\xbe\xad\xde\xef\xbe\xad\xdeA'
```

pcstools.packing.little_endian.**unpack8**(*s*)

Unpacks a bytestring into an integer.

The integer is unpacked as an 8-bit, unsigned, little-endian value.

Examples

```
>>> from pcstools.packing.little_endian import unpack8
>>> unpack8(b'A')
65
>>> unpack8(b'\x01')
1
>>> unpack8(b'')
Traceback (most recent call last):
   ...
ValueError: Argument must be of length 1
>>> unpack8(u'A')
Traceback (most recent call last):
   ...
ValueError: Argument must be a bytestring
```

pcstools.packing.little_endian.**unpack16**(*s*)

Unpacks a bytestring into an integer.

The integer is unpacked as a 16-bit, unsigned, little-endian value.

Examples

```
>>> from pcstools.packing.little_endian import unpack16
>>> unpack16(b'AB')
16961
>>> unpack16(b'\x01\x00')
1
>>> unpack16(b'\x00\x01')
256
>>> unpack16(b'')
Traceback (most recent call last):
```

(continues on next page)

(continued from previous page)

```
   ...
ValueError: Argument must be of length 2
>>> unpack16(u'AB')
Traceback (most recent call last):
   ...
ValueError: Argument must be a bytestring
```

pcstools.packing.little_endian.**unpack32**(*s*)

Unpacks a bytestring into an integer.

The integer is unpacked as a 32-bit, unsigned, little-endian value.

Examples

```
>>> from pcstools.packing.little_endian import unpack32
>>> unpack32(b'ABCD')
1145258561
>>> unpack32(b'\x01\x00\x00\x00')
1
>>> unpack32(b'\x00\x00\x00\x01')
16777216
>>> unpack32(b'')
Traceback (most recent call last):
   ...
ValueError: Argument must be of length 4
>>> unpack32(u'ABCD')
Traceback (most recent call last):
   ...
ValueError: Argument must be a bytestring
```

pcstools.packing.little_endian.**unpack64**(*s*)

Unpacks a bytestring into an integer.

The integer is unpacked as a 64-bit, unsigned, little-endian value.

Examples

```
>>> from pcstools.packing.little_endian import unpack64
>>> unpack64(b'ABCDEFGH')
5208208757389214273
>>> unpack64(b'\x01\x00\x00\x00\x00\x00\x00\x00')
1
>>> unpack64(b'\x00\x00\x00\x00\x00\x00\x00\x01')
72057594037927936
>>> unpack64(b'')
Traceback (most recent call last):
   ...
ValueError: Argument must be of length 8
>>> unpack64(u'ABCDEFGH')
Traceback (most recent call last):
   ...
ValueError: Argument must be a bytestring
```

pcstools.packing.little_endian.**unpack8_many**(*s*)

Unpacks a bytestring into a number of integers.

The integers are unpacked as 8-bit, unsigned, little-endian values.

Examples

```
>>> from pcstools.packing.little_endian import unpack8_many
>>> unpack8_many(b'A')
[65]
>>> unpack8_many(b'\x01')
[1]
>>> unpack8_many(b'')
[]
>>> unpack8_many(b'ABC')
[65, 66, 67]
>>> unpack8_many(u'A')
Traceback (most recent call last):
   ...
ValueError: Argument must be a bytestring
```

pcstools.packing.little_endian.**unpack16_many**(*s*)

Unpacks a bytestring into a number of integers.

The integers are unpacked as 16-bit, unsigned, little-endian values.

Examples

```
>>> from pcstools.packing.little_endian import unpack16_many
>>> unpack16_many(b'AA')
[16705]
>>> unpack16_many(b'\x01\x00')
[1]
>>> unpack16_many(b'')
[]
>>> unpack16_many(b'AABB')
[16705, 16962]
>>> unpack16_many(u'A')
Traceback (most recent call last):
   ...
ValueError: Argument must be a bytestring
>>> unpack16_many(b'A')
Traceback (most recent call last):
   ...
ValueError: Argument must be divisible into groups of 2 bytes
```

pcstools.packing.little_endian.**unpack32_many**(*s*)

Unpacks a bytestring into a number of integers.

The integers are unpacked as 32-bit, unsigned, little-endian values.

Examples

```
>>> from pcstools.packing.little_endian import unpack32_many
>>> unpack32_many(b'AAAA')
[1094795585]
>>> unpack32_many(b'\x01\x00\x00\x00')
[1]
>>> unpack32_many(b'')
[]
>>> unpack32_many(b'AAAABBBB')
[1094795585, 1111638594]
>>> unpack32_many(u'A')
Traceback (most recent call last):
   ...
ValueError: Argument must be a bytestring
>>> unpack32_many(b'A')
Traceback (most recent call last):
   ...
ValueError: Argument must be divisible into groups of 4 bytes
```

pcstools.packing.little_endian.**unpack64_many**(*s*)

Unpacks a bytestring into a number of integers.

The integers are unpacked as 64-bit, unsigned, little-endian values.

Examples

```
>>> from pcstools.packing.little_endian import unpack64_many
>>> unpack64_many(b'AAAAAAAA')
[4702111234474983745]
>>> unpack64_many(b'\x01\x00\x00\x00\x00\x00\x00\x00')
[1]
>>> unpack64_many(b'')
[]
>>> unpack64_many(b'AAAAAAAABBBBBBBB')
[4702111234474983745, 4774451407313060418]
>>> unpack64_many(u'A')
Traceback (most recent call last):
   ...
ValueError: Argument must be a bytestring
>>> unpack64_many(b'A')
Traceback (most recent call last):
   ...
ValueError: Argument must be divisible into groups of 8 bytes
```

1.2 pcstools.packing.big_endian — Packing and unpacking of big-endian integers

pcstools.packing.big_endian.**pack8**(*arg*)

Packs a thing into a bytestring, while recursively decending into iterables.

Integers are packed as 8-bit, unsigned, big-endian values.

Byte-strings and byte-arrays are packed as-is.

Unicode-strings are utf-8 encoded.

Examples

```
>>> from pcstools.packing.big_endian import pack8
>>> assert pack8(0x41) == b'A'
>>> assert pack8([0x41, 0x42]) == b'AB'
>>> assert pack8([0x41, 0x42, (u'hello', b'hi', bytearray(b'sup'))]) == b
↪'ABhellohisup'
>>> assert pack8(u'') == b'\xe2\x98\x83'
>>> pack8(-1)
Traceback (most recent call last):
  ...
ValueError: Number must be positive and below 2**8, but number == -1
>>> # It is also possible to combine pack8 with other packers
>>> from pcstools.packing.big_endian import pack32
>>> assert pack8([0x41, 0x42, pack32(0xdeadbeef)]) == b'AB\xde\xad\xbe\xef'
```

pcstools.packing.big_endian.**pack16**(*arg*)

Packs a thing into a bytestring, while recursively decending into iterables.

Integers are packed as 16-bit, unsigned, big-endian values.

Byte-strings and byte-arrays are packed as-is.

Unicode-strings are utf-8 encoded.

Examples

```
>>> from pcstools.packing.big_endian import pack16
>>> assert pack16(0x4241) == b'BA'
>>> assert pack16([0x4241, 0x4142]) == b'BAAB'
>>> assert pack16([0x4241, (u'hello', b'hi', bytearray(b'sup'))]) == b
↪'BAhellohisup'
>>> assert pack16(u'') == b'\xe2\x98\x83'
>>> pack16(-1)
Traceback (most recent call last):
  ...
ValueError: Number must be positive and below 2**16, but number == -1
>>> # It is also possible to combine pack16 with other packers
>>> from pcstools.packing.big_endian import pack32
>>> assert pack16([0x4241, pack32(0xdeadbeef)]) == b'BA\xde\xad\xbe\xef'
```

pcstools.packing.big_endian.**pack32**(*arg*)

Packs a thing into a bytestring, while recursively decending into iterables.

Integers are packed as 32-bit, unsigned, big-endian values.

Byte-strings and byte-arrays are packed as-is.

Unicode-strings are utf-8 encoded.

Examples

```
>>> from pcstools.packing.big_endian import pack32
>>> assert pack32(0x44434241) == b'DCBA'
>>> assert pack32([0x44434241, 0x41424344]) == b'DCBAABCD'
>>> assert pack32([0x44434241, (u'hello', b'hi', bytearray(b'sup'))]) == b
↪'DCBAhellohisup'
>>> assert pack32(u'') == b'\xe2\x98\x83'
>>> pack32(-1)
Traceback (most recent call last):
   ...
ValueError: Number must be positive and below 2**32, but number == -1
>>> # It is also possible to combine pack32 with other packers
>>> from pcstools.packing.big_endian import pack8
>>> assert pack32([0xdeadbeef, pack8(0x41)]) == b'\xde\xad\xbe\xefA'
```

pcstools.packing.big_endian.**pack64**(*arg*)

Packs a thing into a bytestring, while recursively decending into iterables.

Integers are packed as 64-bit, unsigned, big-endian values.

Byte-strings and byte-arrays are packed as-is.

Unicode-strings are utf-8 encoded.

Examples

```
>>> from pcstools.packing.big_endian import pack64
>>> assert pack64(0x4847464544434241) == b'HGFEDCBA'
>>> assert pack64([0x4847464544434241, 0x4142434445464748]) == b'HGFEDCBAABCDEFGH'
>>> assert pack64([0x4847464544434241, (u'hello', b'hi', bytearray(b'sup'))]) == b
↪'HGFEDCBAhellohisup'
>>> assert pack64(u'') == b'\xe2\x98\x83'
>>> pack64(-1)
Traceback (most recent call last):
   ...
ValueError: Number must be positive and below 2**64, but number == -1
>>> # It is also possible to combine pack64 with other packers
>>> from pcstools.packing.big_endian import pack8
>>> assert pack64([0xdeadbeefdeadbeef, pack8(0x41)]) == b
↪'\xde\xad\xbe\xef\xde\xad\xbe\xefA'
```

pcstools.packing.big_endian.**unpack8**(*s*)

Unpacks a bytestring into an integer.

The integer is unpacked as an 8-bit, unsigned, big-endian value.

Examples

```
>>> from pcstools.packing.big_endian import unpack8
>>> unpack8(b'A')
65
>>> unpack8(b'\x01')
1
>>> unpack8(b'')
```

(continues on next page)

(continued from previous page)

```
Traceback (most recent call last):
   ...
ValueError: Argument must be of length 1
>>> unpack8(u'A')
Traceback (most recent call last):
   ...
ValueError: Argument must be a bytestring
```

pcstools.packing.big_endian.**unpack16**(*s*)

Unpacks a bytestring into an integer.

The integer is unpacked as a 16-bit, unsigned, big-endian value.

Examples

```
>>> from pcstools.packing.big_endian import unpack16
>>> unpack16(b'AB')
16706
>>> unpack16(b'\x01\x00')
256
>>> unpack16(b'\x00\x01')
1
>>> unpack16(b'')
Traceback (most recent call last):
   ...
ValueError: Argument must be of length 2
>>> unpack16(u'AB')
Traceback (most recent call last):
   ...
ValueError: Argument must be a bytestring
```

pcstools.packing.big_endian.**unpack32**(*s*)

Unpacks a bytestring into an integer.

The integer is unpacked as a 32-bit, unsigned, big-endian value.

Examples

```
>>> from pcstools.packing.big_endian import unpack32
>>> unpack32(b'ABCD')
1094861636
>>> unpack32(b'\x01\x00\x00\x00')
16777216
>>> unpack32(b'\x00\x00\x00\x01')
1
>>> unpack32(b'')
Traceback (most recent call last):
   ...
ValueError: Argument must be of length 4
>>> unpack32(u'ABCD')
Traceback (most recent call last):
   ...
ValueError: Argument must be a bytestring
```

pcstools.packing.big_endian.**unpack64**(*s*)

Unpacks a bytestring into an integer.

The integer is unpacked as a 64-bit, unsigned, big-endian value.

Examples

```
>>> from pcstools.packing.big_endian import unpack64
>>> unpack64(b'ABCDEFGH')
4702394921427289928
>>> unpack64(b'\x01\x00\x00\x00\x00\x00\x00\x00')
72057594037927936
>>> unpack64(b'\x00\x00\x00\x00\x00\x00\x00\x01')
1
>>> unpack64(b'')
Traceback (most recent call last):
   ...
ValueError: Argument must be of length 8
>>> unpack64(u'ABCDEFGH')
Traceback (most recent call last):
   ...
ValueError: Argument must be a bytestring
```

pcstools.packing.big_endian.**unpack8_many**(*s*)

Unpacks a bytestring into a number of integers.

The integers are unpacked as 8-bit, unsigned, little-endian values.

Examples

```
>>> from pcstools.packing.big_endian import unpack8_many
>>> unpack8_many(b'A')
[65]
>>> unpack8_many(b'\x01')
[1]
>>> unpack8_many(b'')
[]
>>> unpack8_many(b'ABC')
[65, 66, 67]
>>> unpack8_many(u'A')
Traceback (most recent call last):
   ...
ValueError: Argument must be a bytestring
```

pcstools.packing.big_endian.**unpack16_many**(*s*)

Unpacks a bytestring into a number of integers.

The integers are unpacked as 16-bit, unsigned, little-endian values.

Examples

```
>>> from pcstools.packing.big_endian import unpack16_many
>>> unpack16_many(b'AA')
```

(continues on next page)

(continued from previous page)

```
[16705]
>>> unpack16_many(b'\x00\x01')
[1]
>>> unpack16_many(b'')
[]
>>> unpack16_many(b'AABB')
[16705, 16962]
>>> unpack16_many(u'A')
Traceback (most recent call last):
   ...
ValueError: Argument must be a bytestring
>>> unpack16_many(b'A')
Traceback (most recent call last):
   ...
ValueError: Argument must be divisible into groups of 2 bytes
```

pcstools.packing.big_endian.**unpack32_many**(*s*)

Unpacks a bytestring into a number of integers.

The integers are unpacked as 32-bit, unsigned, little-endian values.

Examples

```
>>> from pcstools.packing.big_endian import unpack32_many
>>> unpack32_many(b'AAAA')
[1094795585]
>>> unpack32_many(b'\x00\x00\x00\x01')
[1]
>>> unpack32_many(b'')
[]
>>> unpack32_many(b'AAAABBBB')
[1094795585, 1111638594]
>>> unpack32_many(u'A')
Traceback (most recent call last):
   ...
ValueError: Argument must be a bytestring
>>> unpack32_many(b'A')
Traceback (most recent call last):
   ...
ValueError: Argument must be divisible into groups of 4 bytes
```

pcstools.packing.big_endian.**unpack64_many**(*s*)

Unpacks a bytestring into a number of integers.

The integers are unpacked as 64-bit, unsigned, little-endian values.

Examples

```
>>> from pcstools.packing.big_endian import unpack64_many
>>> unpack64_many(b'AAAAAAAA')
[4702111234474983745]
>>> unpack64_many(b'\x00\x00\x00\x00\x00\x00\x00\x01')
[1]
```

(continues on next page)

(continued from previous page)

```
>>> unpack64_many(b'')
[]
>>> unpack64_many(b'AAAAAAAABBBBBBBB')
[4702111234474983745, 4774451407313060418]
>>> unpack64_many(u'A')
Traceback (most recent call last):
   ...
ValueError: Argument must be a bytestring
>>> unpack64_many(b'A')
Traceback (most recent call last):
   ...
ValueError: Argument must be divisible into groups of 8 bytes
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

P

U