EzStruct Documentation

Release 0.1.0

Matthew Sachs

January 15, 2014

Contents

Expressive syntax for working with binary data formats and network protocols. Like the `struct` module, but with a more readable syntax, especially if your format has:

- Length-prefixed variable-length byte sequences or strings
- Count-prefixed variable-count repeated fields
- Terminated (null or otherwise) strings
- String encodings
- Numbers which represent enumeration members

Example:

```
tcp = ezstruct.Struct(
    "NET_ENDIAN",
    ezstruct.Field("UINT16", name="sport"),
    ezstruct.Field("UINT16", name="dport"),
    ezstruct.Field("UINT32", name="seqno"),
    ezstruct.Field("UINT32", name="ackno"),
    ezstruct.Field("UINT16",
                   name="flags",
                   value_transform=ezstruct.FieldTransform(
                       pack_flags_bitfield,
                       unpack_flags_bitfield)),
    ezstruct.Field("UINT16", name="window_size"),
    ezstruct.Field("UINT16", name="checksum"),
    ezstruct.Field("UINT16", name="urg"),
    ezstruct.Field("BYTES",
                   name="options",
                   default_pack_value={},
                   length=lambda data: data["offset"] - 5,
                   value_transform=ezstruct.FieldTransform(
                       pack_and_pad_options,
                       unpack_options)))

header_data = {"sport": 123,
               "dport": 456,
               "seqno": 1,
               "ackno": 0,
               "flags": {"offset": 5,
                         "syn": 1},
               "window_size": 100,
               "checksum": 0,
               "urg": 0}
header_bytes = tcp.pack_bytes(header_data)
assert tcp.unpack_bytes(header_bytes) == header_data
```

CHAPTER 1

API

1.1 Delimiter

class `ezstruct.`**`Delimiter`**(*delimiter*)

A fixed-value boundary marker denoting the end of a string.

Common delimiters include `b"\"`, `b","`, and `b"\n"` .

Args: `delimiter`: The delimiter. A `bytes` of length 1.

1.2 Field

class `ezstruct.`**`Field`**(*ft*, *name=None*, *repeat=None*, *default_pack_value=None*, *string_encoding=None*, *string_encoding_errors_policy='strict'*, *length=None*, *value_transform=None*)

A value within a `ezstruct.Struct`.

Args:

`ft`: The field type. See `ezstruct.field_type` for possible values.

`name`: The key to use for the field value in the dictionary used to represent an unpacked version of the structure. If `None`, the value will not be represented in the dict.

`repeat`:

If non-None, the packed structure contains multiple instances of the field, one after the other. This can be:

- An `int`, if there are a fixed number of repetitions; or,
- A `Field` if there are a value number; e.g., if the packed structure contains an 8-byte value indicating the number of copies of this field that follow. For variable repeats, or fixed repeats > 1, the data in the packed structure will be a list containing the value from each repetition.

`default_pack_value`: If non-`None`, the specified value will be used when packing this field if the input to the pack function doesn't contain this field.

`string_encoding`: For string fields, the encoding (as per `codecs`) to use for the string.

`string_encoding_errors_policy`: See `codecs`.

`length`: For variable-length fields (`"STRING"` and `"BYTES"`), the length of the field. This can be:

- An `int`, for fixed-length fields;

- A `Field`, for variable-length fields, e.g., if the packed structure contains an 8-byte value indicating the length of the string that follows.
- A `ezstruct.Delimiter`, for fields whose end is indicated by a specific byte; or,
- A function that takes a single argument, a dictionary of the structure's unpacked data. You can use this for fields whose length is determined by the value of a prior field. When packing, the function is called with the dictionary given to the `pack` function, and if the field's length doesn't match the value returned by the function, `InconsistentLength` is raised. When unpacking, the function is called with a dictionary containing the data unpacked so far.

`value_transform`: See `ezstruct.FieldTransform`.

1.2.1 Field Types

Values for a field's type. Possible values:

Boolean:

"BOOL" A one-byte type; `\x00` unpacks to `False`, all other values unpack to `True`.

Bytes:

"BYTES" A sequence of raw bytes. Fields of this type must have `length` specified.

Integers:

"SINT8" 8 bits, signed.

"SINT16" 16 bits, signed.

"SINT32" 32 bits, signed.

"SINT64" 64 bits, signed.

"UINT8" 8 bits, unsigned.

"UINT16" 16 bits, unsigned.

"UINT32" 32 bits, unsigned.

"UINT64" 64 bits, unsigned.

Floating-Point Numbers:

"FLOAT" 4 bytes in IEEE 754 binary32 format.

"DOUBLE" 8 bytes in IEEE 754 binary64 format.

Strings:

"STRING" A string. Fields of this type must have both `length` and `string_encoding` specified.

1.3 FieldTransform

class `ezstruct.`**`FieldTransform`**(*pack_fn*, *unpack_fn*)

A transformation to apply.

Consists of a pair of functions, each of which is passed a value as its argument and must return a new value to use in its place. You can use this to have a field whose "unpacked" value is something other than the literal value the packed bytes represent.

For instance, if you have a `UINT8` field where the number actually indicates a color, you might have something like:

```
class Color(Enum):
   red = 1
   blue = 2
   green = 3

...
ezstruct.Field(
    "UINT8",
    name="color",
    value_transform=ezstruct.FieldTransform(
        lambda color: color.value,
        lambda color_num: Color(color_num))),
...
```

Args:

pack_fn: A callable to invoke just after reading the value to pack, before doing e.g. string encoding.

unpack_fn: A callable to invoke just before setting the value in the unpack dict, after doing e.g. string decoding.

1.4 Struct

class `ezstruct.`**`Struct`**(*order*, **fields*)

A definition of a binary format.

A `Struct` can be *packed*, converted from a dict describing the data to a sequence of bytes, and *unpacked*, converting from bytes to dict.

Args:

order: Byte order for multi-byte numeric fields. See `ezstruct.byte_order` for possible values.

`fields`: List of `ezstruct.Field`.

pack(*data*, *buf*)

Serialize `data` into an IO buffer.

Args: `data`: The data to pack. `buf`: An `io` buffer to write the packed data to.

pack_bytes(*data*)

Serialize `data` into a new `bytes`.

Args: `data`: The data to pack.

Returns: A `bytes` containing the packed representation of `data`.

unpack(*buf*)

Unserialize data from an IO buffer.

Args: `buf`: An `io` buffer containing data to unpack.

Returns: A dict containing the unpacked data.

unpack_bytes(*the_bytes*)

Unserialize data from a `bytes`.

Args: `the_bytes`: The byte sequence to unpack.

Returns: A dict containing the unpacked data.

1.4.1 Byte Orders

Values for a structure's byte ordering. Possible values:

"NATIVE_ENDIAN" Host byte ordering.

"LITTLE_ENDIAN" Little-endian, e.g. the 2-byte number `"\x00\x01"` is `1`.

"BIG_ENDIAN" Big-endian, e.g. the 2-byte number `"\x01\x00"` is `1`.

"NET_ENDIAN" Network byte order, i.e. big-endian.

CHAPTER 2

About

Created and maintained by Matthew Sachs.

The source repository for this package can be found at GitHub.

CHAPTER 3

Special Thanks

- Google, Inc.

CHAPTER 4

Further Reading

4.1 History

v0.1.0, 2014-01-15 Initial release.

4.2 Authors

- Matthew Sachs

CHAPTER 5

Indices and tables

- *genindex*
- *modindex*
- *search*

e