Temporenc

Release 0.1.0

November 07, 2014

Contents

This is a Python library implementing the temporenc format for dates and times.

Features:

- Support for all *temporenc* types
- Interoperability with the `datetime` module
- Time zone support, including conversion to local time
- Compatibility with both Python 2 (2.6+) and Python 3 (3.2+)
- Decent performance
- Permissive BSD license

Contents

CHAPTER 1

Installation

Use `pip` to install the library (e.g. into a `virtualenv`):

```
$ pip install temporenc
```

CHAPTER 2

Usage

2.1 Basic usage

All functionality is provided by a single module with the name `temporenc`:

```
>>> import temporenc
```

To encode date and time information into a byte string, use the `packb()` function:

```
>>> temporenc.packb(year=2014, month=10, day=23)
b'\x8f\xbd6'
```

This function automatically determines the most compact representation for the provided information. In this case, the result uses *temporenc* type `D`, but if you want to use a different type, you can provide it explicitly:

```
>>> temporenc.packb(type='DT', year=2014, month=10, day=23)
b'\x1fzm\xff\xff'
```

To unpack a byte string, use `unpackb()`:

```
>>> moment = temporenc.unpackb(b'\x1fzm\xff\xff')
>>> moment
<temporenc.Moment '2014-10-23'>
>>> print(moment)
2014-10-23
```

As you can see, unpacking returns a `Moment` instance. This class has a reasonable string representation, but it is generally more useful to access the individual components using one of its many attributes:

```
>>> print(moment.year)
2014
>>> print(moment.month)
10
>>> print(moment.day)
13
>>> print(moment.second)
None
```

Since all fields are optional in *temporenc* values, and since no time information was set in this example, some of the attributes (e.g. *second*) are *None*.

2.2 Integration with the `datetime` module

Python has built-in support for date and time handling, provided by the `datetime` module in the standard library, which is how applications usually work with date and time information. Instead of specifying all the fields manually when packing data, which is cumbersome and error-prone, the `temporenc` module integrates with the built-in `datetime` module:

```
>>> import datetime
>>> now = datetime.datetime.now()
>>> now
datetime.datetime(2014, 10, 23, 18, 45, 23, 612883)
>>> temporenc.packb(now)
b'W\xde\x9bJ\xd5\xe5hL'
```

As you can see, instead of specifying all the components manually, instances of the built-in `datetime.datetime` class can be passed directly as the first argument to `packb()`. This also works for `datetime.date` and `datetime.time` instances.

Since the Python `datetime` module *always* uses microsecond precision, this library defaults to *temporenc* types with sub-second precision (e.g. `DTS`) when an instance of one of the `datetime` classes is passed. If no subsecond precision is required, you can specify a different type to save space:

```
>>> temporenc.packb(now, type='DT')
b'\x1fzm+W'
```

The integration with the `datetime` module works both ways. Instances of the `Moment` class (as returned by the unpacking functions) can be converted to the standard date and time classes using the `datetime()`, `date()`, and `time()` methods:

```
>>> moment = temporenc.unpackb(b'W\xde\x9bJ\xd5\xe5hL')
>>> moment
<temporenc.Moment '2014-10-23 18:45:23.612883'>
>>> moment.datetime()
datetime.datetime(2014, 10, 23, 18, 45, 23, 612883)
>>> moment.date()
datetime.date(2014, 10, 23)
>>> moment.time()
datetime.time(18, 45, 23, 612883)
```

Conversion to and from classes from the `datetime` module have full time zone support. See the API docs for `Moment.datetime()` for more details about time zone handling.

> **Warning:** The Python `temporenc` module only concerns itself with encoding and decoding. It does *not* do any date and time calculations, and hence does not validate that dates are correct. For example, it handles the non-existent date *February 30* just fine. Always convert to native classes from the `datetime` module if you need to work with date and time information in your application.

2.3 Working with file-like objects

The *temporenc* encoding format allows for reading data from a stream without knowing in advance how big the encoded byte string is. This library supports this through the `unpack()` function, which consumes exactly the required number of bytes from the stream:

```
>>> import io
>>> fp = io.BytesIO()  # this could be a real file
```

```
>>> fp.write(b'W\xde\x9bJ\xd5\xe5hL')
>>> fp.write(b'foo')
>>> fp.seek(0)
>>> temporenc.unpack(fp)
<temporenc.Moment '2014-10-23 18:45:23.612883'>
>>> fp.tell()
8
>>> fp.read()
b'foo'
```

For writing directly to a file-like object, the `pack()` function can be used, though this is just a shortcut.

CHAPTER 3

API

The `packb()` and `unpackb()` functions operate on byte strings.

`temporenc.`**`packb`**(*value=None*, *type=None*, *year=None*, *month=None*, *day=None*, *hour=None*, *minute=None*, *second=None*, *millisecond=None*, *microsecond=None*, *nanosecond=None*, *tz_offset=None*)

Pack date and time information into a byte string.

If specified, *value* must be a `datetime.datetime`, `datetime.date`, or `datetime.time` instance.

The *type* specifies the *temporenc* type to use. Valid types are `D`, `T`, `DT`, `DTZ`, `DTS`, or `DTSZ`. If not specified, the most compact encoding that can represent the provided information will be determined automatically. Note that instances of the classes in the `datetime` module always use microsecond precision, so make sure to specify a more compact type if no sub-second precision is required.

Most applications would only use the *value* and *type* arguments; the other arguments allow for encoding data that does not fit the conceptual date and time model used by the standard library's `datetime` module.

Note: Applications that require lexicographical ordering of encoded values should always explicitly specify a type to use.

All other arguments can be used to specify individual pieces of information that make up a date or time. If both *value* and more specific fields are provided, the individual fields override the values extracted from *value*, e.g. `packb(datetime.datetime.now(), minute=0, second=0)` encodes the start of the current hour.

The sub-second precision arguments (*millisecond*, *microsecond*, and *nanosecond*) must not be used together, since those are conceptually mutually exclusive.

Note: The *value* argument is the only positional argument. All other arguments *must* be specified as keyword arguments (even though this is not enforced because of Python 2 compatibility).

Parameters

- **value** – instance of one of the `datetime` classes (optional)
- **type** (*str*) – *temporenc* type (optional)
- **year** (*int*) – year (optional)
- **month** (*int*) – month (optional)
- **day** (*int*) – day (optional)
- **hour** (*int*) – hour (optional)

- **minute** (*int*) – minute (optional)
- **second** (*int*) – second (optional)
- **millisecond** (*int*) – millisecond (optional)
- **microsecond** (*int*) – microsecond (optional)
- **nanosecond** (*int*) – nanosecond (optional)
- **tz_offset** (*int*) – time zone offset in minutes from UTC (optional)

Returns encoded *temporenc* value

Return type bytes

`temporenc.`**`unpackb`**(*value*)

Unpack a *temporenc* value from a byte string.

If no valid value could be read, this raises `ValueError`.

Parameters **value** (*bytes*) – a byte string (or *bytearray*) to parse

Returns a parsed *temporenc* structure

Return type `Moment`

The `pack()` and `unpack()` functions operate on file-like objects.

`temporenc.`**`pack`**(*fp*, **args*, ***kwargs*)

Pack date and time information and write it to a file-like object.

This is a short-hand for writing a packed value directly to a file-like object. There is no additional behaviour. This function only exists for API parity with the `unpack()` function.

Except for the first argument (the file-like object), all arguments (both positional and keyword) are passed on to `packb()`. See `packb()` for more information.

Parameters

- **fp** (*file-like*) – writeable file-like object
- **args** – propagated to `packb()`
- **kwargs** – propagated to `packb()`

Returns number of bytes written

Return type int

`temporenc.`**`unpack`**(*fp*)

Unpack a *temporenc* value from a file-like object.

This function consumes exactly the number of bytes required to unpack a single *temporenc* value.

If no valid value could be read, this raises `ValueError`.

Parameters **fp** (*file-like*) – readable file-like object

Returns a parsed *temporenc* structure

Return type `Moment`

Both `unpackb()` and `unpack()` return an instance of the `Moment` class.

class `temporenc.`**`Moment`**(*year*, *month*, *day*, *hour*, *minute*, *second*, *nanosecond*, *tz_offset*)

Container to represent a parsed *temporenc* value.

Each constituent part is accessible as an instance attribute. These are: `year`, `month`, `day`, `hour`, `minute`, `second`, `millisecond`, `microsecond`, `nanosecond`, and `tz_offset`. Since *temporenc* allows partial date and time information, any attribute can be `None`.

The attributes for sub-second precision form a group that is either completely empty (all attributes are `None`) or completely filled (no attribute is `None`).

This class is intended to be a read-only immutable data structure; assigning new values to attributes is not supported.

Instances are hashable and can be used as dictionary keys or as members of a set. Instances representing the same moment in time have the same hash value. Time zone information is not taken into account for hashing purposes, since time zone aware values must have their constituent parts in UTC.

Instances of this class can be compared to each other, with earlier dates sorting first. As with hashing, time zone information is not taken into account, since the actual data must be in UTC in those cases.

Note: This class must not be instantiated directly; use one of the unpacking functions like `unpackb()` instead.

`date`(*strict=True*, *local=False*)

Convert this value to a `datetime.date` instance.

See the documentation for the `datetime()` method for more information.

Parameters

- **strict** (*bool*) – whether to use strict conversion rules
- **local** (*bool*) – whether to convert to local time

Returns converted value

Type *datetime.date*

`datetime`(*strict=True*, *local=False*)

Convert this value to a `datetime.datetime` instance.

Since the classes in the `datetime` module do not support missing values, this will fail when one of the required components is not set, which is indicated by raising a `ValueError`.

The default is to perform a strict conversion. To ease working with partial dates and times, the *strict* argument can be set to *False*. In that case this method will try to convert the value anyway, by substituting a default value for any missing component, e.g. a missing time is set to *00:00:00*. Note that these substituted values are bogus and should not be used for any application logic, but at least this allows applications to use things like `.strftime()` on partial dates and times.

The *temporenc* format allows inclusion of a time zone offset. Date and time information in the *temporenc* types `DTZ` and `DTSZ` is always stored as UTC, but the original UTC offset is included, which makes conversion to the original local time possible. When converting to a `datetime` instance, time zone information is handled as follows:

- When no time zone information was present in the original data (e.g. when unpacking *temporenc* type `DT`), the return value will be a naive *datetime* instance, i.e. its `tzinfo` attribute is *None*.
- If the original data did include time zone information, the return value will be a time zone aware instance. No conversion to local time is performed by default, which means the instance will have a `tzinfo` attribute corresponding to UTC. This means time zone information will be lost, and the return value will be in UTC.

  If this is not desired, i.e. the application wants to access the original local time, set the *local* argument to *True*. In that case the data will be converted to local time, and the return value will have a `tzinfo` attribute corresponding to the time zone offset.

Parameters

- **strict** (*bool*) – whether to use strict conversion rules
- **local** (*bool*) – whether to convert to local time

Returns converted value

Type *datetime.datetime*

time(*strict=True*, *local=False*)

Convert this value to a `datetime.time` instance.

See the documentation for the `datetime()` method for more information.

Parameters

- **strict** (*bool*) – whether to use strict conversion rules
- **local** (*bool*) – whether to convert to local time

Returns converted value

Type *datetime.date*

CHAPTER 4

Contributing

Source code, including the test suite, is maintained at Github:

> temporenc-python on github

Feel free to submit feedback, report issues, bring up improvement ideas, and contribute fixes!

CHAPTER 5

License

D

M

P

T

U