zulu Documentation

Release 2.0.0

Derrick Gilland

Jun 27, 2021

Contents

A drop-in replacement for native datetimes that embraces UTC

CHAPTER 1

Links

- Project: https://github.com/dgilland/zulu
- Documentation: https://zulu.readthedocs.io
- PyPI: https://pypi.python.org/pypi/zulu/
- Github Actions: https://github.com/dgilland/zulu/actions

CHAPTER 2

Features

- All datetime objects converted and stored as UTC.
- Parses ISO8601 formatted strings and POSIX timestamps by default.
- Timezone representation applied only during string output formatting or when casting to native datetime object.
- Drop-in replacement for native datetime objects.
- Python 3.6+

CHAPTER 3

Quickstart

Install using pip:

```
pip3 install zulu
```

```
import zulu

zulu.now()
# <Zulu [2016-07-25T19:33:18.137493+00:00]>

dt = zulu.parse('2016-07-25T19:33:18.137493+00:00')
# <Zulu [2016-07-25T19:33:18.137493+00:00]>

dt = zulu.create(2016, 7, 25, 19, 33, 18, 137493)
# <Zulu [2016-07-25T19:33:18.137493+00:00]>

dt.isoformat()
# '2016-07-25T19:33:18.137493+00:00'

dt.timestamp()
# 1469475198.137493

dt.naive
# datetime.datetime(2016, 7, 25, 19, 33, 18, 137493)

dt.datetime
# datetime.datetime(2016, 7, 25, 19, 33, 18, 137493, tzinfo=<UTC>)

dt.format('%Y-%m-%d')
# 2016-07-25

dt.format('YYYY-MM-dd')
# 2016-07-25

dt.format("E, MMM d, ''YY")
```

(continues on next page)

(continued from previous page)

```
# "Mon, Jul 25, '16"

dt.format("E, MMM d, ''YY", locale='de')
# "Mo., Juli 25, '16"

dt.format("E, MMM d, ''YY", locale='fr')
# "lun., juil. 25, '16"

dt.shift(hours=-5, minutes=10)
# <Zulu [2016-07-25T14:43:18.137493+00:00]>

dt.replace(hour=14, minute=43)
# <Zulu [2016-07-25T14:43:18.137493+00:00]>

dt.start_of('day')
# <Zulu [2016-07-25T00:00:00+00:00]>

dt.end_of('day')
# <Zulu [2016-07-25T23:59:59.999999+00:00]>

dt.span('hour')
# (<Zulu [2016-07-25T19:00:00+00:00]>, <Zulu [2016-07-25T19:59:59.999999+00:00]>)

dt.time_from(dt.end_of('day'))
# '4 hours ago'

dt.time_to(dt.end_of('day'))
# 'in 4 hours'

list(zulu.range('hour', dt, dt.shift(hours=4)))
# [Zulu [2016-07-25T19:33:18.137493+00:00]>,
#  Zulu [2016-07-25T20:33:18.137493+00:00]>,
#  Zulu [2016-07-25T21:33:18.137493+00:00]>,
#  Zulu [2016-07-25T22:33:18.137493+00:00]>]

list(zulu.span_range('minute', dt, dt.shift(minutes=4)))
# [(Zulu [2016-07-25T19:33:00+00:00]>, Zulu [2016-07-25T19:33:59.999999+00:00]>),
#  (Zulu [2016-07-25T19:34:00+00:00]>, Zulu [2016-07-25T19:34:59.999999+00:00]>),
#  (Zulu [2016-07-25T19:35:00+00:00]>, Zulu [2016-07-25T19:35:59.999999+00:00]>),
#  (Zulu [2016-07-25T19:36:00+00:00]>, Zulu [2016-07-25T19:36:59.999999+00:00]>)]

zulu.parse_delta('1w 3d 2h 32m')
# <Delta [10 days, 2:32:00]>

zulu.parse_delta('2:04:13:02.266')
# <Delta [2 days, 4:13:02.266000]>

zulu.parse_delta('2 days, 5 hours, 34 minutes, 56 seconds')
# <Delta [2 days, 5:34:56]>
```

CHAPTER 4

Why Zulu?

Why zulu instead of native datetimes:

- Zulu has extended datetime features such as `parse()`, `format()`, `shift()`, and python-dateutil timezone support.
- Parses ISO8601 and timestamps by default without any extra arguments.
- Easier to reason about `Zulu` objects since they are only ever UTC datetimes.
- Clear delineation between UTC and other time zones where timezone representation is only applicable for display or conversion to native datetime.
- Supports more string parsing/formatting options using Unicode date patterns as well as `strptime/strftime` directives.

Why zulu instead of Arrow:

- Zulu is a drop-in replacement for native datetimes (inherits from `datetime.datetime`). No need to convert using `arrow.datetime` when you need a datetime (zulu is always a datetime).
- Stricter parsing to avoid silent errors. For example, one might expect `arrow.get('02/08/1987', 'MM/DD/YY')` to fail (input does not match format) but it gladly returns `<Arrow [2019-02-08T00:00:00+00:00)` whereas `zulu.parse('02/08/1987', '%m/%d/%y')` throws `zulu.parser.ParseError: Value "02/08/1987" does not match any format in ['%m/%d/%y']`.
- Avoids timezone/DST shifting bugs by only dealing with UTC datetimes when applying timedeltas or performing other calculations.
- Supports `strptime/strftime` as well as Unicode date patterns for string parsing/formatting.

CHAPTER 5

Special Thanks

Special thanks goes out to the authors/contributors of the following libraries that have made it possible for `zulu` to exist:

- Babel
- iso8601
- python-dateutil
- pytimeparse
- pytz

For the full documentation, please visit https://zulu.readthedocs.io.

CHAPTER 6

Guide

6.1 Installation

zulu requires Python >= 3.6.

To install from PyPI:

```
pip install zulu
```

6.2 User's Guide

Zulu's main type is `zulu.Zulu` which represents a fixed UTC datetime object.

```
import zulu

dt = zulu.parse('2016-07-25T19:33:18.137493+00:00')
# <Zulu [2016-07-25T19:33:18.137493+00:00]>

assert isinstance(dt, zulu.Zulu)

from datetime import datetime
assert isinstance(dt, datetime)
```

It's a drop-in replacement for native datetime objects (it inherits from `datetime.datetime`) but deals only with UTC time zones internally.

6.2.1 Basic Data Access

All the attributes and methods from datetime are available along with a few new ones:

```
assert dt.year == 2016
assert dt.month == 7
assert dt.day == 25
assert dt.hour == 19
assert dt.minute == 33
assert dt.second == 18
assert dt.microsecond == 137493
assert dt.tzname() == 'UTC'

dt.utcoffset()
# datetime.timedelta(0)

dt.dst()
# datetime.timedelta(0)

dt.isoformat()
# '2016-07-25T19:33:18.137493+00:00'

dt.weekday()
# 0

dt.isoweekday()
# 1

dt.isocalendar()
# (2016, 30, 1)

dt.ctime()
# 'Mon Jul 25 19:33:18 2016'

dt.toordinal()
# 736170

dt.timetuple()
# time.struct_time(tm_year=2016, tm_mon=7, tm_mday=25, tm_hour=19, tm_min=33, tm_
↪sec=18,
#                  tm_wday=0, tm_yday=207, tm_isdst=0)

dt.utctimetuple()
# time.struct_time(tm_year=2016, tm_mon=7, tm_mday=25, tm_hour=19, tm_min=33, tm_
↪sec=18,
#                  tm_wday=0, tm_yday=207, tm_isdst=0)

dt.timestamp()
# 1469475198.137493

dt.date()
# datetime.date(2016, 7, 25)

dt.time()
# datetime.time(19, 33, 18, 137493)

dt.timetz()
# datetime.time(19, 33, 18, 137493, tzinfo=<UTC>)
```

Along with a few new ones:

```
dt.naive
# datetime.datetime(2016, 7, 25, 19, 33, 18, 137493)

dt.datetime
# datetime.datetime(2016, 7, 25, 19, 33, 18, 137493, tzinfo=<UTC>)

dt.is_leap_year()
# True

dt.days_in_month()
# 31

tuple(dt)
# (2016, 7, 25, 19, 33, 18, 137493, <UTC>)
```

6.2.2 Parsing and Formatting

By default, `zulu.parse` will look for either an ISO8601 formatted string or a POSIX timestamp while assuming a UTC timezone when no explicit timezone found in the string:

```
zulu.parse('2016-07-25 15:33:18-0400')
# <Zulu [2016-07-25T19:33:18+00:00]>

zulu.parse('2016-07-25 15:33:18-0400', zulu.ISO8601)
# <Zulu [2016-07-25T19:33:18+00:00]>

zulu.parse('2016-07-25')
# <Zulu [2016-07-25T00:00:00+00:00]>

zulu.parse('2016-07-25 19:33')
# <Zulu [2016-07-25T19:33:00+00:00]>

zulu.parse(1469475198.0)
# <Zulu [2016-07-25T19:33:18+00:00]>

zulu.parse(1469475198.0, zulu.TIMESTAMP)
# <Zulu [2016-07-25T19:33:18+00:00]>
```

Multiple formats can be supplied and `zulu.parse` will try them all:

```
zulu.parse('3/2/1992', 'ISO8601')
# zulu.parser.ParseError: Value "3/2/1992" does not match any format in "ISO8601"
# (Unable to parse date string '3/2/1992')

dt = zulu.parse('3/2/1992', ['ISO8601', 'MM/dd/YYYY'])
# <Zulu [1992-03-02T00:00:00+00:00]>
```

As shown above, special parse format keywords are supported. See *Keyword Parse Formats* for details.

Other time zones can be substituted for naive datetimes by setting `default_tz`:

```
zulu.parse('2016-07-25', default_tz='US/Eastern')
# <Zulu [2016-07-25T04:00:00+00:00]>

zulu.parse('2016-07-25', default_tz='local')
# <Zulu [2016-07-25T04:00:00+00:00]>
```

The default timezone is ignored when the input has it set:

```
zulu.parse('2016-07-25T15:33:18-0700', default_tz='US/Eastern')
# <Zulu [2016-07-25T22:33:18+00:00]>
```

String parsing/formatting in `Zulu` supports both strftime/strptime directives and Unicode date patterns.

```
dt.format('%Y-%m-%d %H:%M:%S%z')
# '2016-07-25 19:33:18+0000'

dt.format('YYYY-MM-dd HH:mm:ssZ')
# '2016-07-25 19:33:18+0000'

dt.format('%Y-%m-%d %H:%M:%S%z', tz='US/Eastern')
# '2016-07-25 15:33:18-0400'

dt.format('%Y-%m-%d %H:%M:%S%z', tz='local')
# '2016-07-25 15:33:18-0400'

zulu.parse('2016-07-25 15:33:18-0400', '%Y-%m-%d %H:%M:%S%z')
# <Zulu [2016-07-25T19:33:18+00:00]>
```

You can even use `zulu.parser.format_datetime` with native datetimes:

```
from zulu.parser import UTC, format_datetime

native = datetime(2016, 7, 25, 19, 33, 18, 137493, tzinfo=UTC)

format_datetime(native, '%Y-%m-%d %H:%M:%S%z')
# '2016-07-25 19:33:18+0000'

format_datetime(native, 'YYYY-MM-dd HH:mm:ssZ')
# '2016-07-25 19:33:18+0000'

dt = Zulu.fromdatetime(native)
format_datetime(dt, 'YYYY-MM-dd HH:mm:ssZ')
# '2016-07-25 19:33:18+0000'
```

Keyword Parse Formats

The following keywords can be supplied to `zulu.parse` in place of a format directive or pattern:

```
zulu.parse(1469475198, 'timestamp')
# <Zulu [2016-07-25T19:33:18+00:00]>
```

Keyword	Description	Sample Input
ISO8601	Parse ISO8601 string	• 2016-07-25 15:33:18-0400 • 2016-07-25 15:33 • 2016-07-25 • 2016-07
timestamp	Parse POSIX timestamp	• 1469475198 • 1469475198.314218

6.2.3 Format Tokens

Zulu supports two different styles of string parsing/formatting tokens:

- All Python strptime/strftime directives
- A subset of Unicode date patterns

Either style can be used during parsing:

```
dt = zulu.parse('07/25/16 15:33:18 -0400', '%m/%d/%y %H:%M:%S %z')
# <Zulu [2016-07-25T19:33:18+00:00]>

dt = zulu.parse('07/25/16 15:33:18 -0400', 'MM/dd/YY HH:mm:ss Z')
# <Zulu [2016-07-25T19:33:18+00:0
```

and formatting:

```
dt.format('%m/%d/%y %H:%M:%S %z')
# '07/25/16 19:33:18 +0000'

dt.format('MM/dd/YY HH:mm:ss Z')
'07/25/16 19:33:18 +0000'
```

Format Directives

All directives from https://docs.python.org/3.5/library/datetime.html#strftime-and-strptime-behavior are supported.

Date Patterns

A subset of patterns from http://www.unicode.org/reports/tr35/tr35-19.html#Date_Field_Symbol_Table are supported for parsing while _all_ patterns are supported for formatting:

Attribute	Style	Pattern	Examples
Year	4-digit	YYYY	2000, 2001, 2002 … 2015, 2016
Year	2-digit	YY	00, 01, 02 … 15, 16
Month	full name	MMMM	January, February, March
Month	abbr name	MMM	Jan, Feb, Mar … Nov, Dec
Month	int, padded	MM	01, 02, 03 … 11, 12
Month	int, no padding	M	1, 2, 3 … 11, 12
Day of Month	int, padded	dd	01, 02, 03 … 30, 31
Day of Month	int, no padding	d	1, 2, 3 … 30, 31
Day of Year	int, padded	DDD	001, 002, 003 … 054, 055 … 364, 365
Day of Year	int, padded	DD	01, 02, 03 … 54, 55 … 364, 365
Day of Year	int, no padding	D	1, 2, 3 … 54, 55 … 364, 365
Weekday	full name	EEEE	Sunday, Monday, Tuesday … Friday, Saturday
Weekday	abbr name	EEE	Sun, Mon, Tue … Fri, Sat
Weekday	abbr name	EE	Sun, Mon, Tue … Fri, Sat
Weekday	abbr name	E	Sun, Mon, Tue … Fri, Sat
Weekday	abbr name	eee	Sun, Mon, Tue … Fri, Sat
Weekday	int, padded	ee	01, 02, 03 … 06, 07
Weekday	int, no padding	e	1, 2, 3 … 6, 7
Hour	24h, padded	HH	00, 01, 02 … 22, 23

Continued on next page

Table 1 – continued from previous page

Attribute	Style	Pattern	Examples
Hour	24h, no padding	H	0, 1, 2 … 22, 23
Hour	12h, padded	hh	00, 01, 02 … 11, 12
Hour	12h, no padding	h	0, 1, 2, … 11, 12
AM / PM	upper case	a	AM, PM
Minute	int, padded	mm	00, 01, 02 … 58, 59
Minute	int, no padding	m	0, 1, 2 … 58, 59
Second	int, padded	ss	00, 01, 02 … 58, 59
Second	int, no padding	s	0, 1, 2 … 58, 59
Microsecond	int, padded	SSSSSS	000000, 000001 … 999998, 999999
Microsecond	int, truncated	SSSSS	00000, 00001 … 99998, 99999
Microsecond	int, truncated	SSSS	0000, 0001 … 9998, 9999
Microsecond	int, truncated	SSS	000, 001 … 998, 999
Microsecond	int, truncated	SS	00, 01 … 98, 99
Microsecond	int, truncated	S	0, 1 … 8, 9
Timezone	w/o separator	Z	-1100, -1000 … +0000 … +1100, +1200

Humanization

You can humanize the difference between two `Zulu` objects with `Zulu.time_from` and `Zulu.time_to`:

```
dt
# <Zulu [2016-07-25T19:33:18.137493+00:00]>

dt.time_from(dt.end_of_day())
# '4 hours ago'

dt.time_to(dt.end_of_day())
# 'in 4 hours'

dt.time_from(dt.start_of_day())
# 'in 20 hours'

dt.time_to(dt.start_of_day())
# '20 hours ago'

zulu.now()
# <Zulu [2016-08-12T04:16:17.007335+00:00]>

dt.time_from_now()
# 2 weeks ago

dt.time_to_now()
# in 2 weeks
```

6.2.4 Time Zone Handling

Time zones other than UTC are not expressable within a `Zulu` instance. Other time zones are only ever applied when either converting a `Zulu` object to a native datetime (via `Zulu.astimezone`) or during string formatting (via `Zulu.format`). `Zulu` understands both `tzinfo` objects and IANA Timezone Database string names (also known as the Olson database).

```
local = dt.astimezone()
# same as doing dt.astimezone('local')
# datetime.datetime(2016, 7, 25, 15, 33, 18, 137493,
#                   tzinfo=tzlocal())

pacific = dt.astimezone('US/Pacific')
# datetime.datetime(2016, 7, 25, 12, 33, 18, 137493,
#                   tzinfo=tzfile('/usr/share/zoneinfo/US/Pacific'))

import pytz
mountain = dt.astimezone(pytz.timezone('US/Mountain'))
# datetime.datetime(2016, 7, 25, 13, 33, 18, 137493,
#                   tzinfo=<DstTzInfo 'US/Mountain' MDT-1 day, 18:00:00 DST>)
```

6.2.5 Shifting, Replacing, and Copying

Zulu can easily apply timedelta's using the `shift` method:

```
shifted = dt.shift(hours=-5, minutes=10)
# <Zulu [2016-07-25T14:43:18.137493+00:00]>

assert shifted is not dt
```

And add and subtract with the `add` and `subtract` methods:

```
shifted = dt.subtract(hours=5).add(minutes=10)
# <Zulu [2016-07-25T14:43:18.137493+00:00]>

# First argument to subtract() can be a timedelta or dateutil.relativedelta
shifted = dt.subtract(timedelta(hours=5))
# <Zulu [2016-07-25T14:33:18+00:00]>

# First argument to subtract() can also be another datetime object
dt.subtract(shifted)
# datetime.timedelta(0, 18000)

# First argument to add() can be a timedelta or dateutil.relativedelta
dt.add(timedelta(minutes=10))
# <Zulu [2016-07-25T19:43:18+00:00]>
```

Or replace datetime attributes:

```
replaced = dt.replace(hour=14, minute=43)
# <Zulu [2016-07-25T14:43:18.137493+00:00]>

assert replaced is not dt
```

Or even make a copy:

```
copied = dt.copy()
# <Zulu [2016-07-25T19:33:18.137493+00:00]>

assert copied is not dt
assert copied == dt
```

Note: Since `Zulu` is meant to be immutable, both `shift`, `replace`, and `copy` return new `Zulu` instances while leaving the original instance unchanged.

6.2.6 Spans, Ranges, Starts, and Ends

You can get the span across a time frame:

```
dt = Zulu(2015, 4, 4, 12, 30, 37, 651839)

dt.span('century')
# (<Zulu [2000-01-01T00:00:00+00:00]>, <Zulu [2099-12-31T23:59:59.999999+00:00]>)

dt.span('decade')
# (<Zulu [2010-01-01T00:00:00+00:00]>, <Zulu [2019-12-31T23:59:59.999999+00:00]>)

dt.span('year')
# (<Zulu [2015-01-01T00:00:00+00:00]>, <Zulu [2015-12-31T23:59:59.999999+00:00]>)

dt.span('month')
# (<Zulu [2015-04-01T00:00:00+00:00]>, <Zulu [2015-04-30T23:59:59.999999+00:00]>)

dt.span('week')
# (<Zulu [2015-03-30T00:00:00+00:00]>, <Zulu [2015-04-05T23:59:59.999999+00:00]>)

dt.span('day')
# (<Zulu [2015-04-04T00:00:00+00:00]>, <Zulu [2015-04-04T23:59:59.999999+00:00]>)

dt.span('hour')
# (<Zulu [2015-04-04T12:00:00+00:00]>, <Zulu [2015-04-04T12:59:59.999999+00:00]>)

dt.span('minute')
# (<Zulu [2015-04-04T12:30:00+00:00]>, <Zulu [2015-04-04T12:30:59.999999+00:00]>)

dt.span('second')
# (<Zulu [2015-04-04T12:30:37+00:00]>, <Zulu [2015-04-04T12:30:37.999999+00:00]>)

dt.span('century', count=3)
# (<Zulu [2000-01-01T00:00:00+00:00]>, <Zulu [2299-12-31T23:59:59.999999+00:00]>)

dt.span('decade', count=3)
# (<Zulu [2010-01-01T00:00:00+00:00]>, <Zulu [2039-12-31T23:59:59.999999+00:00]>)
```

Or you can get just the start or end of a time frame:

```
dt.start_of('day')  # OR dt.start_of_day()
# <Zulu [2015-04-04T00:00:00+00:00]>

dt.end_of('day')  # OR dt.end_of_day()
# <Zulu [2015-04-04T23:59:59.999999+00:00]>

dt.end_of('year', count=3)  # OR dt.end_of_year()
# <Zulu [2017-12-31T23:59:59.999999+00:00]>
```

Note: Supported time frames are `century`, `decade`, `year`, `month`, `week`, `day`, `hour`,

`minute`, `second` and are accessible both from `start_of(frame)`/`end_of(frame)` and `start_of_<frame>()`/`end_of_<frame>`.

You can get a range of time spans:

```
start = Zulu(2015, 4, 4, 12, 30)
end = Zulu(2015, 4, 4, 16, 30)

for span in zulu.span_range('hour', start, end):
    print(span)
# (<Zulu [2015-04-04T12:00:00+00:00]>, <Zulu [2015-04-04T12:59:59.999999+00:00]>)
# (<Zulu [2015-04-04T13:00:00+00:00]>, <Zulu [2015-04-04T13:59:59.999999+00:00]>)
# (<Zulu [2015-04-04T14:00:00+00:00]>, <Zulu [2015-04-04T14:59:59.999999+00:00]>)
# (<Zulu [2015-04-04T15:00:00+00:00]>, <Zulu [2015-04-04T15:59:59.999999+00:00]>)
```

Or you can iterate over a range of datetimes:

```
start = Zulu(2015, 4, 4, 12, 30)
end = Zulu(2015, 4, 4, 16, 30)

for dt in zulu.range('hour', start, end):
    print(dt)
# <Zulu [2015-04-04T12:30:00+00:00]>
# <Zulu [2015-04-04T13:30:00+00:00]>
# <Zulu [2015-04-04T14:30:00+00:00]>
```

Note: Supported range/span time frames are `century`, `decade`, `year`, `month`, `week`, `day`, `hour`, `minute`, `second`.

6.2.7 Time Deltas

In addition to having a drop-in replacement for `datetime`, zulu also has a drop-in replacement for `timedelta`:

```
delta = zulu.parse_delta('1w 3d 2h 32m')
# <Delta [10 days, 2:32:00]>

assert isinstance(delta, zulu.Delta)

from datetime import timedelta
assert isinstance(delta, timedelta)

zulu.parse_delta('2:04:13:02.266')
# <Delta [2 days, 4:13:02.266000]>

zulu.parse_delta('2 days, 5 hours, 34 minutes, 56 seconds')
# <Delta [2 days, 5:34:56]>
```

Other formats that `zulu.parse_delta` can parse are:

- `32m`
- `2h32m`
- `3d2h32m`
- `1w3d2h32m`

- `1w 3d 2h 32m`
- `1 w 3 d 2 h 32 m`
- `4:13`
- `4:13:02`
- `4:13:02.266`
- `2:04:13:02.266`
- `2 days, 4:13:02` (`uptime` format)
- `2 days, 4:13:02.266`
- `5hr34m56s`
- `5 hours, 34 minutes, 56 seconds`
- `5 hrs, 34 mins, 56 secs`
- `2 days, 5 hours, 34 minutes, 56 seconds`
- `1.2 m`
- `1.2 min`
- `1.2 mins`
- `1.2 minute`
- `1.2 minutes`
- `172 hours`
- `172 hr`
- `172 h`
- `172 hrs`
- `172 hour`
- `1.24 days`
- `5 d`
- `5 day`
- `5 days`
- `5.6 wk`
- `5.6 week`
- `5.6 weeks`

Similar to `Zulu.time_to/from`, `Delta` objects can be humanized with the `Delta.format` method:

```
delta = zulu.parse_delta('2h 32m')
# <Delta [2:32:00]>

delta.format()
# '3 hours'

delta.format(add_direction=True)
# 'in 3 hours'
```

(continues on next page)

(continued from previous page)

```
zulu.parse_delta('-2h 32m').format(add_direction=True)
# '3 hours ago'

delta.format(granularity='day')
# '1 day'

delta.format(locale='de')
# '3 Stunden'

delta.format(locale='fr', add_direction=True)
# 'dans 3 heures'

delta.format(threshold=0)
# '0 years'

delta.format(threshold=0.1)
# '0 days'

delta.format(threshold=0.2)
# '3 hours'

delta.format(threshold=5)
# '152 minutes'

delta.format(threshold=155)
# '9120 seconds'

delta.format(threshold=155, granularity='minute')
# '152 minutes'

delta.format(style='long')
# '3 hours'

delta.format(style='short')
# '3 hr'

delta.format(style='narrow')
# '3h'
```

6.2.8 Utilities

zulu.to_seconds

Easily convert time units from microseconds to weeks into seconds:

```
zulu.to_seconds(seconds=5, minutes=2, hours=3, days=2, weeks=1)
# 788525

zulu.to_seconds(milliseconds=25300, seconds=5, minutes=2)
# 150.3
```

zulu.Timer

A timer object that can be used for keeping track of elapsed time or as a coutdown timer.

As a timer:

```
timer = zulu.Timer()
timer.start()

# is the same as...
timer = zulu.Timer().start()

timer.started()
# True

timer.stopped()
# False

timer.elapsed()
# 0.0003867149353027344

timer.stop()

timer.elapsed()
# 0.0009307861328125
```

Can be used as a context manager to track the duration of blocks of code:

```
timer = zulu.Timer()

with timer:
    time.sleep(1)

# is the same as ...
# with zulu.Timer() as timer:
#     ...

timer.elapsed()
# 1.001131534576416

# can be used multiple times to accumulate durations
with timer:
    time.sleep(2)

timer.elapsed()
# 3.0032811164855957

# reset the timer
timer.reset()

timer.started()
# False

timer.elapsed()
# 0
```

And as a countdown timer:

```
# timer that runs out after 15 seconds
timer = zulu.Timer(timeout=15)
timer.start()

timer.done()
# False

timer.remaining()
# 14.999720811843872

time.sleep(5)

timer.remaining()
# 9.99406123161316

time.sleep(10)

timer.done()
# True

timer.remaining()
# -0.01725912094116211

# restart the timer by calling start() again
timer.start()

timer.done()
# False
```

6.3 API Reference

zulu.**now**()
Alias to *Zulu.now()*.

zulu.**create**(*year=1970*, *month=1*, *day=1*, *hour=0*, *minute=0*, *second=0*, *microsecond=0*, *tzinfo=None*, *, *fold=0*)
Factory function to create a new *Zulu* datetime using the same arguments as the class.

Parameters

- **year** (*int|dict*) – Date year `1 <= year <= 9999` or `dict` containing keys corresponding to initialization parameter names.
- **month** (*int*) – Date month `1 <= month <= 12`.
- **day** (*int*) – Date day `1 <= day <= number of days in the given month and year`
- **hour** (*int, optional*) – Time hour `0 <= hour < 24`. Defaults to `0`.
- **minute** (*int, optional*) – Time minute `0 <= minute < 60`. Defaults to `0`.
- **second** (*int, optional*) – Time second `0 <= second < 60`. Defaults to `0`.
- **microsecond** (*int, optional*) – Time microsecond `0 <= microsecond < 1000000`. Defaults to `0`.

- **tzinfo** (*None|str|tzinfo, optional*) – Timezone information as either a `str` or `tzinfo` subclass. If value is a `str`, it will be converted to a `dateutil.tz` timezone. If value is `None`, the datetime values given are assumed to in UTC. Defaults to `None`.

`zulu.`**`parse`**(*obj*, *formats=None*, *default_tz=None*)

Return *Zulu* object parsed from *obj*.

Parameters

- **obj** (*mixed*) – Object to parse into a *Zulu* object.
- **formats** (*str|list, optional*) – List of string formats to use when parsing. Defaults to `["ISO8601", "timestamp"]`.
- **default_tz** (*None|str|tzinfo, optional*) – Default timezone to use when parsed datetime object does not contain a timezone. Defaults to `UTC`.

Returns *Zulu*

`zulu.`**`range`**(*frame*, *start*, *end*)

Yield ranges of *Zulu* instances from *start* to *end* in steps of time frame, *frame*.

Parameters

- **frame** (*str*) – A time frame (e.g. year, month, day, minute, etc).
- **start** (*datetime*) – The starting datetime.
- **end** (*datetime*) – The ending datetime.

Yields *Zulu* – Datetime values ranging from the given start and end datetimes.

`zulu.`**`span_range`**(*frame*, *start*, *end*)

Return a range of time spans from *start* to *end* in steps of time frame, *frame*.

Parameters

- **frame** (*str*) – A time frame (e.g. year, month, day, minute, etc).
- **start** (*datetime*) – The starting datetime.
- **end** (*datetime*) – The ending datetime.

Yields *tuple* – 2-element tuple of Zulu time spans

`zulu.`**`parse_delta`**(*obj*)

Return *Delta* object parsed from *obj*.

Parameters **obj** (*str|number|timedelta*) – Object to parse into a *Delta* object.

Returns *Delta*

class `zulu.`**`Zulu`**

The Zulu class represents an immutable UTC datetime object. Any timezone information given to it during instantiation results in the datetime object being converted from that timezone to UTC. If timezone information is not given, then it's assumed the datetime is a UTC value. All arithmetic is perform on the underlying UTC datetime object. Zulu has no concept of timezone shifting in this regard. Instead, localization occurs only when formatting a Zulu object as a string.

The Zulu class is a drop-in replacement for a native datetime object, but does not represent itself in any time zone other than UTC.

Parameters

- **year** (*int|dict*) – Date year `1 <= year <= 9999` or `dict` containing keys corresponding to initialization parameter names.

- **month** (*int*) – Date month `1 <= month <= 12`.
- **day** (*int*) – Date day `1 <= day <= number of days in the given month and year`
- **hour** (*int, optional*) – Time hour `0 <= hour < 24`. Defaults to `0`.
- **minute** (*int, optional*) – Time minute `0 <= minute < 60`. Defaults to `0`.
- **second** (*int, optional*) – Time second `0 <= second < 60`. Defaults to `0`.
- **microsecond** (*int, optional*) – Time microsecond `0 <= microsecond < 1000000`. Defaults to `0`.
- **tzinfo** (*None|str|tzinfo, optional*) – Timezone information as either a `str` or `tzinfo` subclass. If value is a `str`, it will be converted to a `dateutil.tz` timezone. If value is `None`, the datetime values given are assumed to in UTC. Defaults to `None`.

add(*other=None*, *years=0*, *months=0*, *weeks=0*, *days=0*, *hours=0*, *minutes=0*, *seconds=0*, *microseconds=0*)

Add time using a timedelta created from the supplied arguments and return a new *Zulu* instance. The first argument can be a *:class:'timedelta* or `dateutil.relativedelta` object in which case all other arguments are ignored and the object is added to this datetime.

Parameters

- **other** (*timedelta|relativedelta, optional*) – A `timedelta` or `dateutil.relativedelta` object to add.
- **years** (*int*) – Years to add.
- **months** (*int*) – Months to add.
- **weeks** (*int*) – Weeks to add.
- **days** (*int*) – Days to add.
- **hours** (*int*) – Hours to add.
- **minutes** (*int*) – Minutes to add.
- **seconds** (*int*) – Seconds to add.
- **microseconds** (*int*) – Microseconds to add.

Returns *Zulu*

astimezone(*tz='local'*)

Return datetime shifted to timezone *tz*.

Note: This returns a native datetime object.

Parameters **tz** (*None|str|tzinfo, optional*) – Timezone to shift to. Defaults to *"local"* which uses the local timezone.

Returns *datetime*

classmethod combine(*date*, *time*)

Return *Zulu* object by combining the date part from *date* and the time part from *time*.

Parameters

- **date** (*mixed*) – Either a *Zulu*, `datetime.date`, or `datetime.datetime` object to use as the date part.

- **date** – Either a *Zulu* or `datetime.time` object to use as the time part.

Returns *Zulu*

copy()
Return a new :class‘Zulu‘ instance with the same datetime value.

Returns *Zulu*

ctime()
Return ctime() style string.

date()
Return date object with same year, month and day.

datetime
The Zulu object as a native datetime.

Note: This returns a native datetime object.

Returns *datetime*

datetimetuple()
Return datetime `tuple` containing `(year, month, day, hour, minute, second, microsecond, tzinfo)`.

Returns tuple

datetuple()
Return date `tuple` containing `(year, month, day)`.

Returns tuple

days_in_month()
Return the number of days in the month.

Returns int

dst()
Return self.tzinfo.dst(self).

end_of(*frame*, *count=1*)
Return the end of a given time frame for this datetime.

Parameters

- **frame** (*str*) – A time frame (e.g. year, month, day, minute, etc).
- **count** (*int*) – Number of frames to span.

Returns *Zulu*

end_of_century(*count=1*)
Return a new *Zulu* set to the end of the century of this datetime.

Parameters **count** (*int*) – Number of frames to span.

Returns *Zulu*

end_of_day(*count=1*)
Return a new *Zulu* set to the end of the day of this datetime.

Parameters **count** (*int*) – Number of frames to span.

Returns *Zulu*

end_of_decade(*count=1*)
Return a new *Zulu* set to the end of the decade of this datetime.

Parameters **count** (*int*) – Number of frames to span.

Returns *Zulu*

end_of_hour(*count=1*)
Return a new *Zulu* set to the end of the hour of this datetime.

Parameters **count** (*int*) – Number of frames to span.

Returns *Zulu*

end_of_minute(*count=1*)
Return a new *Zulu* set to the end of the minute of this datetime.

Parameters **count** (*int*) – Number of frames to span.

Returns *Zulu*

end_of_month(*count=1*)
Return a new *Zulu* set to the end of the month of this datetime.

Parameters **count** (*int*) – Number of frames to span.

Returns *Zulu*

end_of_second(*count=1*)
Return a new *Zulu* set to the end of the second of this datetime.

Parameters **count** (*int*) – Number of frames to span.

Returns *Zulu*

end_of_week(*count=1*)
Return a new *Zulu* set to the end of the week of this datetime. uses ISO8601 definition of week: week start is monday.

Parameters **count** (*int*) – Number of frames to span.

Returns *Zulu*

end_of_year(*count=1*)
Return a new *Zulu* set to the end of the year of this datetime.

Parameters **count** (*int*) – Number of frames to span.

Returns *Zulu*

format(*format=None*, *tz=None*, *locale='en_US_POSIX'*)
Return datetime as a string using the format string *format* while optionally converting to timezone *tz* first.

Note: A `Locale` object or string identifier can be provided to display the object in that particular locale **but only when using date pattern tokens.** Using a locale other than the current system locale is not supported for strftime tokens.

Parameters

- **format** (*str*) – Format to return string in. If `None`, ISO 8601 format is used. Defaults to `None`.

- **tz** (*None|str|tzinfo, optional*) – Timezone to convert to before formatting. Defaults to None.
- **locale** (*str|Locale, optional*) – A Locale object or locale identifier. Defaults to system default.

Returns str

classmethod fromdatetime(*dt*)
Return *Zulu* object from a native datetime object.

Returns *Zulu*

classmethod fromgmtime(*struct*)
Return *Zulu* object from a tuple like that returned by time.gmtime that represents a UTC datetime.

Parameters **struct** (*tuple*) – Tuple like that returned by time.gmtime.

Returns *Zulu*

fromisoformat()
string -> datetime from datetime.isoformat() output

classmethod fromlocaltime(*struct*)
Return *Zulu* object from a tuple like that returned by time.localtime that represents a local datetime.

Parameters **struct** (*tuple*) – Tuple like that returned by time.localtime.

Returns *Zulu*

classmethod fromordinal(*ordinal*)
Return *Zulu* object from a proleptic Gregorian ordinal, where January 1 of year 1 has ordinal 1.

Parameters **ordinal** (*int*) – A proleptic Gregorian ordinal.

Returns *Zulu*

classmethod fromtimestamp(*timestamp*, *tz=tzutc()*)
Return *Zulu* object from a POSIX timestamp.

Parameters

- **timestamp** (*int*) – POSIX timestamp such as is returned by time.time().
- **tz** (*UTC*) – This argument is ignored and always set to UTC. It is present only for datetime class compatibility.

Returns *Zulu*

is_after(*other*)
Return whether this datetime is after *other*.

Parameters **other** (*datetime*) – Other datetime to compare.

Returns bool

is_before(*other*)
Return whether this datetime is before *other*.

Parameters **other** (*datetime*) – Other datetime to compare.

Returns bool

is_between(*start*, *end*)
Return whether this datetime is between *start* and *end* inclusively.

Parameters

- **start** (*datetime*) – Starting datetime to compare.
- **end** (*datetime*) – Ending datetime to compare.

Returns bool

is_leap_year()
Return whether this datetime's `year` is a leap year.

Returns bool

is_on_or_after(*other*)
Return whether this datetime is after or on *other*.

Parameters **other** (*datetime*) – Other datetime to compare.

Returns bool

is_on_or_before(*other*)
Return whether this datetime is before or on *other*.

Parameters **other** (*datetime*) – Other datetime to compare.

Returns bool

isocalendar()
Return a 3-tuple containing ISO year, week number, and weekday.

isoformat()
[sep] -> string in ISO 8601 format, YYYY-MM-DDT[HH[:MM[:SS[.mmm[uuu]]]]][+HH:MM]. sep is used to separate the year from the time, and defaults to 'T'. timespec specifies what components of the time to include (allowed values are 'auto', 'hours', 'minutes', 'seconds', 'milliseconds', and 'microseconds').

isoweekday()
Return the day of the week represented by the date. Monday == 1 ... Sunday == 7

naive
The datetime object as a naive datetime (tzinfo=None).

Note: This returns a native datetime object.

Returns *Zulu*

classmethod now()
Return the current UTC date and time as *Zulu* object.

Returns *Zulu*

classmethod parse(*obj*, *formats=None*, *default_tz=None*)
Return *Zulu* object parsed from *obj*.

Parameters

- **obj** (*mixed*) – Object to parse into a *Zulu* object.
- **formats** (*str|list, optional*) – List of string formats to use when parsing. Defaults to `["ISO8601", "timestamp"]`.
- **default_tz** (*None|str|tzinfo, optional*) – Default timezone to use when parsed datetime object does not contain a timezone. Defaults to `UTC`.

Returns *Zulu*

classmethod range(*frame*, *start*, *end*)
Yield ranges of *Zulu* instances from *start* to *end* in steps of time frame, *frame*.

Parameters

- **frame** (*str*) – A time frame (e.g. year, month, day, minute, etc).
- **start** (*datetime*) – The starting datetime.
- **end** (*datetime*) – The ending datetime.

Yields *Zulu* – Datetime values ranging from the given start and end datetimes.

replace(*year=None*, *month=None*, *day=None*, *hour=None*, *minute=None*, *second=None*, *microsecond=None*, *tzinfo=None*, ***, *fold=None*)
Replace datetime attributes and return a new *Zulu* instance.

Parameters

- **year** (*None|int, optional*) – Year to replace.
- **month** (*None|int, optional*) – Month to replace.
- **week** (*None|int, optional*) – Week to replace.
- **day** (*None|int, optional*) – Day to replace.
- **hour** (*None|int, optional*) – Hour to replace.
- **minute** (*None|int, optional*) – Minute to replace.
- **second** (*None|int, optional*) – Second to replace.
- **microsecond** (*None|int, optional*) – Microsecond to replace.
- **tzinfo** (*None|str|tzinfo, optional*) – Timezone to replace.
- **fold** (*None|int, optional*) – Fold to replace.

Returns *Zulu*

shift(*other=None*, *years=0*, *months=0*, *weeks=0*, *days=0*, *hours=0*, *minutes=0*, *seconds=0*, *microseconds=0*)
Shift datetime forward or backward using a timedelta created from the supplied arguments and return a new *Zulu* instance.

Parameters

- **other** (*timedelta|relativedelta, optional*) – A `timedelta` or `dateutil.relativedelta` object to add.
- **years** (*int*) – Years to shift.
- **months** (*int*) – Months to shift.
- **weeks** (*int*) – Weeks to shift.
- **days** (*int*) – Days to shift.
- **hours** (*int*) – Hours to shift.
- **minutes** (*int*) – Minutes to shift.
- **seconds** (*int*) – Seconds to shift.
- **microseconds** (*int*) – Microseconds to shift.

Returns *Zulu*

span (*frame*, *count=1*)
Returns two new *Zulu* objects corresponding to the time span between this object and the given time frame.

Parameters

- **frame** (*str*) – A time frame (e.g. year, month, day, minute, etc).
- **count** (*int*) – Number of frames to span.

Returns (*start_of_frame*, *end_of_frame*)

Return type tuple

classmethod span_range (*frame*, *start*, *end*)
Return a range of time spans from *start* to *end* in steps of time frame, *frame*.

Parameters

- **frame** (*str*) – A time frame (e.g. year, month, day, minute, etc).
- **start** (*datetime*) – The starting datetime.
- **end** (*datetime*) – The ending datetime.

Yields *tuple* – 2-element tuple of Zulu time spans

start_of (*frame*)
Return the start of the given time frame for this datetime.

Parameters **frame** (*str*) – A time frame (e.g. year, month, day, minute, etc).

Returns *Zulu*

start_of_century ()
Return a new *Zulu* set to the start of the century of this datetime.

Returns *Zulu*

start_of_day ()
Return a new *Zulu* set to the start of the day of this datetime.

Returns *Zulu*

start_of_decade ()
Return a new *Zulu* set to the start of the decade of this datetime.

Returns *Zulu*

start_of_hour ()
Return a new *Zulu* set to the start of the hour of this datetime.

Returns *Zulu*

start_of_minute ()
Return a new *Zulu* set to the start of the minute of this datetime.

Returns *Zulu*

start_of_month ()
Return a new *Zulu* set to the start of the month of this datetime.

Returns *Zulu*

start_of_second ()
Return a new *Zulu* set to the start of the second of this datetime.

Returns *Zulu*

start_of_week()
Return a new *Zulu* set to the start of the week of this datetime. uses ISO8601 definition of week: week start is monday.

Returns *Zulu*

start_of_year()
Return a new *Zulu* set to the start of the year of this datetime.

Returns *Zulu*

strftime()
format -> strftime() style string.

strptime()
string, format -> new datetime parsed from a string (like time.strptime()).

subtract(*other=None*, *years=0*, *months=0*, *weeks=0*, *days=0*, *hours=0*, *minutes=0*, *seconds=0*, *microseconds=0*)
Subtract time using a timedelta created from the supplied arguments and return a new *Zulu* instance. The first argument can be a *Zulu*, *datetime*, `timedelta`, or `dateutil.relativedelta` object in which case all other arguments are ignored and the object is subtracted from this datetime.

Parameters

- **other** (*datetime|timedelta|relativedelta, optional*) – A `datetime`, `timedelta`, or `dateutil.relativedelta` object to subtract.
- **years** (*int*) – Years to subtract.
- **months** (*int*) – Months to subtract.
- **weeks** (*int*) – Weeks to subtract.
- **days** (*int*) – Days to subtract.
- **hours** (*int*) – Hours to subtract.
- **minutes** (*int*) – Minutes to subtract.
- **seconds** (*int*) – Seconds to subtract.
- **microseconds** (*int*) – Microseconds to subtract.

Returns if subtracting `timedelta` or `timedelta`. `timedelta`: if subtracting a *datetime* or *Zulu*.

Return type *Zulu*

time()
Return time object with same time but with tzinfo=None.

time_from(*dt*, *format='long'*, *granularity='second'*, *threshold=0.85*, *add_direction=True*, *locale=None*)
Return "time ago" difference between this datetime and another as a humanized string.

Parameters

- **dt** (*datetime*) – A datetime object.
- **format** (*str, optional*) – Can be one of "long", "short", or "narrow". Defaults to *"long"*.
- **granularity** (*str, optional*) – The smallest unit that should be displayed. The value can be one of "year", "month", "week", "day", "hour", "minute" or "second". Defaults to *"second"*.

- **threshold** (*float, optional*) – Factor that determines at which point the presentation switches to the next higher unit. Defaults to *0.85*.
- **add_direction** (*bool, optional*) – If `True` the return value will include directional information (e.g. *'1 hour ago'*, *'in 1 hour'*). Defaults to `False`.
- **locale** (*str|Locale, optional*) – A `Locale` object or locale identifier. Defaults to system default.

Returns str

time_from_now(*format='long'*, *granularity='second'*, *threshold=0.85*, *add_direction=True*, *locale=None*)

Return "time ago" difference between this datetime and now as a humanized string.

Parameters

- **format** (*str, optional*) – Can be one of "long", "short", or "narrow". Defaults to *"long"*.
- **granularity** (*str, optional*) – The smallest unit that should be displayed. The value can be one of "year", "month", "week", "day", "hour", "minute" or "second". Defaults to *"second"*.
- **threshold** (*float, optional*) – Factor that determines at which point the presentation switches to the next higher unit. Defaults to *0.85*.
- **add_direction** (*bool, optional*) – If `True` the return value will include directional information (e.g. *'1 hour ago'*, *'in 1 hour'*). Defaults to `False`.
- **locale** (*str|Locale, optional*) – A `Locale` object or locale identifier. Defaults to system default.

Returns str

time_to(*dt*, *format='long'*, *granularity='second'*, *threshold=0.85*, *add_direction=True*, *locale=None*)

Return "time to" difference between another datetime and this one as a humanized string.

Parameters

- **dt** (*datetime*) – A datetime object.
- **format** (*str, optional*) – Can be one of "long", "short", or "narrow". Defaults to *"long"*.
- **granularity** (*str, optional*) – The smallest unit that should be displayed. The value can be one of "year", "month", "week", "day", "hour", "minute" or "second". Defaults to *"second"*.
- **threshold** (*float, optional*) – Factor that determines at which point the presentation switches to the next higher unit. Defaults to *0.85*.
- **add_direction** (*bool, optional*) – If `True` the return value will include directional information (e.g. *'1 hour ago'*, *'in 1 hour'*). Defaults to `False`.
- **locale** (*str|Locale, optional*) – A `Locale` object or locale identifier. Defaults to system default.

Returns str

time_to_now(*format='long'*, *granularity='second'*, *threshold=0.85*, *add_direction=True*, *locale=None*)

Return "time to" difference between now and this datetime as a humanized string.

Parameters

- **format** (*str, optional*) – Can be one of “long”, “short”, or “narrow”. Defaults to *“long”*.
- **granularity** (*str, optional*) – The smallest unit that should be displayed. The value can be one of “year”, “month”, “week”, “day”, “hour”, “minute” or “second”. Defaults to *“second”*.
- **threshold** (*float, optional*) – Factor that determines at which point the presentation switches to the next higher unit. Defaults to *0.85*.
- **add_direction** (*bool, optional*) – If `True` the return value will include directional information (e.g. *‘1 hour ago’*, *‘in 1 hour’*). Defaults to `False`.
- **locale** (*str|Locale, optional*) – A `Locale` object or locale identifier. Defaults to system default.

Returns str

timestamp()
Return the POSIX timestamp.

Returns `float`

timetuple()
Return time tuple, compatible with time.localtime().

timetz()
Return time object with same time and tzinfo.

today()
Current date or datetime: same as self.__class__.fromtimestamp(time.time()).

toordinal()
Return proleptic Gregorian ordinal. January 1 of year 1 is day 1.

tzname()
Return self.tzinfo.tzname(self).

classmethod utcfromtimestamp(*timestamp*)
Return *Zulu* object from a POSIX timestamp.

Parameters **timestamp** (*int*) – POSIX timestamp such as is returned by `time.time()`.

Returns *Zulu*

classmethod utcnow()
Return the current UTC date and time as *Zulu* object.

Returns *Zulu*

utcoffset()
Return self.tzinfo.utcoffset(self).

utctimetuple()
Return UTC time tuple, compatible with time.localtime().

weekday()
Return the day of the week represented by the date. Monday == 0 . . . Sunday == 6

class zulu.**Delta**
An extension of `datetime.timedelta` that provides additional functionality.

days
Number of days.

format (*format='long'*, *granularity='second'*, *threshold=0.85*, *add_direction=False*, *locale=None*)
Return timedelta as a formatted string.

Parameters

- **format** (*str, optional*) – Can be one of "long", "short", or "narrow". Defaults to *"long"*.
- **granularity** (*str, optional*) – The smallest unit that should be displayed. The value can be one of "year", "month", "week", "day", "hour", "minute" or "second". Defaults to *"second"*.
- **threshold** (*float, optional*) – Factor that determines at which point the presentation switches to the next higher unit. Defaults to *0.85*.
- **add_direction** (*bool, optional*) – If `True` the return value will include directional information (e.g. *'1 hour ago'*, *'in 1 hour'*). Defaults to `False`.
- **locale** (*str|Locale, optional*) – A `Locale` object or locale identifier. Defaults to system default.

Returns str

classmethod fromtimedelta(*delta*)
Return *Delta* object from a native timedelta object.

Returns *Delta*

microseconds
Number of microseconds (>= 0 and less than 1 second).

classmethod parse(*obj*)
Return *Delta* object parsed from *obj*.

Parameters **obj** (*str|number|timedelta*) – Object to parse into a *Delta* object.

Returns *Delta*

seconds
Number of seconds (>= 0 and less than 1 day).

total_seconds()
Total seconds in the duration.

class zulu.**ParseError**
Exception raised when an object cannot be parsed as a datetime.

6.4 Developer Guide

This guide provides an overview of the tooling this project uses and how to execute developer workflows using the developer CLI.

6.4.1 Python Environments

This Python project is tested against different Python versions. For local development, it is a good idea to have those versions installed so that tests can be run against each.

There are libraries that can help with this. Which tools to use is largely a matter of preference, but below are a few recommendations.

For managing multiple Python versions:

- pyenv
- OS package manager (e.g. apt, yum, homebrew, etc)
- Build from source

For managing Python virtualenvs:

- pyenv-virtualenv
- pew
- python-venv

6.4.2 Tooling

The following tools are used by this project:

Tool	Description	Configuration
black	Code formatter	`pyproject.toml`
isort	Import statement formatter	`setup.cfg`
docformatter	Docstring formatter	`setup.cfg`
flake8	Code linter	`setup.cfg`
pylint	Code linter	`pylintrc`
pytest	Test framework	`setup.cfg`
tox	Test environment manager	`tox.ini`
invoke	CLI task execution library	`tasks.py`

6.4.3 Workflows

The following workflows use developer CLI commands via invoke and are defined in `tasks.py`.

Autoformat Code

To run all autoformatters:

```
inv fmt
```

This is the same as running each autoformatter individually:

```
inv black
inv isort
inv docformatter
```

Lint

To run all linters:

```
inv lint
```

This is the same as running each linter individually:

```
inv flake8
inv pylint
```

Test

To run all unit tests:

```
inv unit
```

To run unit tests and builds:

```
inv test
```

Test on All Supported Python Versions

To run tests on all supported Python versions:

```
tox
```

This requires that the supported versions are available on the PATH.

Build Package

To build the package:

```
inv build
```

This will output the source and binary distributions under `dist/`.

Build Docs

To build documentation:

```
inv docs
```

This will output the documentation under `docs/_build/`.

Serve Docs

To serve docs over HTTP:

```
inv docs -s|--server [-b|--bind 127.0.0.1] [-p|--port 8000]

inv docs -s
inv docs -s -p 8080
inv docs -s -b 0.0.0.0 -p 8080
```

Delete Build Files

To remove all build and temporary files:

```
inv clean
```

This will remove Python bytecode files, egg files, build output folders, caches, and tox folders.

Release Package

To release a new version of the package to https://pypi.org:

```
inv release
```

6.4.4 CI/CD

This project uses Github Actions for CI/CD:

- https://github.com/dgilland/zulu/actions

CHAPTER 7

Project Info

7.1 License

MIT License

Copyright (c) 2020 Derrick Gilland

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.2 Versioning

This project follows Semantic Versioning with the following caveats:

- Only the public API (i.e. the objects imported into the zulu module) will maintain backwards compatibility between MINOR version bumps.
- Objects within any other parts of the library are not guaranteed to not break between MINOR version bumps.

With that in mind, it is recommended to only use or import objects from the main module, zulu.

7.3 Changelog

7.3.1 v2.0.0 (2021-06-26)

- Drop support for Python 3.4 and 3.5.

7.3.2 v1.3.1 (2021-06-26)

- Fix compatibility issue with Python 3.4 and 3.5 by replacing f-string usage with string-format.

7.3.3 v1.3.0 (2021-01-07)

- Add Python 3.9 support.
- Fix bug in `Zulu.time_from`, `Zulu.time_to`, `Zulu.time_from_now`, and `Zulu.time_to_now` where keyword arguments weren't passed to underlying `Delta.format` call.
- Fix bug in `Zulu.format` where "YY" and "YYYY" format patterns would return the year in "Week of Year" based calendars instead of the regular calendar year.

7.3.4 v1.2.0 (2020-01-14)

- Add Python 3.8 support.
- Add `'week'` option to `Zulu.start_of`, `Zulu.end_of`, `Zulu.span`, and `Zulu.span_range`. Thanks ThomasChiroux!
- Fix bug in `Zulu.astimezone` in Python 3.8 due to change in return type from `super().asdatetime`. In Python<=3.7, `super().asdatetime` returned as instance of `datetime`, but in Python 3.8 another instance of `Zulu` was returned instead. `Zulu.astimezone` will now return a `datetime` instance in Python 3.8.

7.3.5 v1.1.1 (2019-08-14)

- Remove unused parameter in `zulu.Timer.__init__()`.

7.3.6 v1.1.0 (2018-11-01)

- Add `fold` attribute support to `Zulu`.
- Add `zulu.to_seconds` for converting units of time to total number of seconds.
- Add `zulu.Timer` class that can be used to track elapsed time (like a stopwatch) or as a countdown timer.

7.3.7 v1.0.0 (2018-08-20)

- Drop support for Python 2.7.

7.3.8 v0.12.1 (2018-07-16)

- Support Python 3.7.

7.3.9 v0.12.0 (2017-07-11)

- Add `Zulu.datetimetuple()`.
- Add `Zulu.datetuple()`.
- Remove `Zulu.__iter__` method. (**breaking change**)
- Remove `Delta.__iter__` method. (**breaking change**)

7.3.10 v0.11.0 (2017-06-28)

- Add Python 3.6 support.
- Add `Delta.__iter__` method that yields 2-element tuples like `Zulu.__iter__`. Delta values are normalized into integer values distributed from the higher units to the lower units.
- Add `Delta.__float__` and `Delta.__int__` methods for converting to seconds.
- Add `Zulu.__float__` and `Zulu.__int__` methods for converting to epoch seconds.
- Fix issue in Python 3.6 where `zulu.now()` returned a naive datetime `Zulu` instance.
- Make `Zulu.__iter__` yield 2-element tuples containing `(unit, value)` like `(('year', 2000), ('month', 12), ...)` so it can be converted to a `dict` with proper keys easier. (**breaking change**)
- Remove previously deprecated `zulu.delta()` function. Use `zulu.parse_delta()` instead. (**breaking change**)
- Rename modules: (**breaking change**)
  - `zulu.datetime -> zulu.zulu`
  - `zulu.timedelta -> zulu.delta`

7.3.11 v0.10.1 (2017-02-15)

- Provide fallback for the default value of `locale` in `Delta.format()` when a locale is not known via environment variables.

7.3.12 v0.10.0 (2017-02-13)

- Add `zulu.parse_delta` as alias for `Delta.parse`.
- Deprecate `zulu.delta` in favor of `zulu.parse_delta`.
- Allow first argument to `Zulu()`, `Zulu.parse()`, and `zulu.parse()` to be a `dict` containing keys corresponding to initialization parameters.
- Fix error message for invalid timezone strings so that the supplied string is shown correctly.

7.3.13 v0.9.0 (2016-11-21)

- Require `python-dateutil>=2.6.0`. (**breaking change**)
- Replace usage of `pytz` timezone handling for strings with `dateutil.tz.gettz`. Continue to support `pytz` timezones during `Zulu()` object creation. (**breaking change**).

- Replace default UTC timezone with `dateutil.tz.tzutc()`. Was previously `pytz.UTC`. (**breaking change**)
- Replace local timezone with `dateutil.tz.tzlocal()`. Was previously set by the `tzlocal` library. (**breaking change**)

7.3.14 v0.8.0 (2016-10-31)

- Add comparison methods to `Zulu`:
  - `is_before`
  - `is_on_or_before`
  - `is_after`
  - `is_on_or_after`
  - `is_between`

7.3.15 v0.7.3 (2016-10-24)

- Optimize `Zulu()` constructor by eliminating multiple unnecessary calls to `datetime` constructor.

7.3.16 v0.7.2 (2016-09-06)

- Fix `Zulu` not being pickle-able.

7.3.17 v0.7.1 (2016-08-22)

- Add missing magic method overrides for `Delta` for `+delta`, `-delta`, and `abs(delta)` so that `Delta` is returned instead of `datetime.timedelta`.
  - `__pos__`
  - `__neg__`
  - `__abs__`

7.3.18 v0.7.0 (2016-08-22)

- Make `Zulu.__sub__` and `Zulu.subtract` return a `Delta` object instead of `datetime.timedelta`.
- Make `zulu.delta` and `Zulu.Delta.parse` accept a number.
- Allow the first argument to `Zulu.shift` be a timedelta or relative delta object.
- Ensure that mathematical magic methods for `Delta` return `Delta` objects and not `datetime.timedelta`.
  - `__add__`
  - `__radd__`
  - `__sub__`
  - `__mul__`
  - `__rmul__`

  - `__floordiv__`
  - `__truediv__` (Python 3 only)
  - `__div__` (Python 2 only)
  - `__mod__` (Python 3 only)
  - `__divmod__` (Python 3 only)

7.3.19 v0.6.0 (2016-08-14)

- Replace internal implementation of Unicode date pattern formatting with Babel's `format_datetime`. **breaking change**
- Remove support for formatting to timestamp using `X` and `XX`. **breaking change**
- Rename parse-from-timestamp token from `X` to `timestamp`. **breaking change**
- Add `zulu.create` as factory function to create a `zulu.Zulu` instance.
- Add locale support to `Zulu.format` when using Unicode date pattern format tokens.
- Restore locale support to `Delta.format`.

7.3.20 v0.5.0 (2016-08-13)

- Remove locale support from `Delta.format`. Locale is currently not supported in `Zulu.format` so decided to disable it in `Delta.format` until both can have it. **breaking change**

7.3.21 v0.4.0 (2016-08-13)

- Rename `zulu.DateTime` to `zulu.Zulu`. **breaking change**
- Rename `Zulu.isleap` to `Zulu.is_leap_year`. **breaking change**
- Remove `zulu.format` alias (function can be accessed at `zulu.parser.format_datetime`). **breaking change**
- Remove `Zulu.leapdays`. **breaking change**
- Add `Zulu.days_in_month`.
- Add `zulu.Delta` class that inherits from `datetime.timedelta`.
- Add `zulu.delta` as alias to `zulu.Delta.parse`.
- Add `Zulu.time_from`, `Zulu.time_to`, `Zulu.time_from_now`, and `Zulu.time_to_now` that return "time ago" or "time to" humanized strings.
- Add `zulu.range` as alias to `Zulu.range`.
- Add `zulu.span_range` as alias to `Zulu.span_range`.
- Make time units (years, months, weeks, days, hours, minutes, seconds, microseconds) keyword arguments only for `Zulu.add/subtract`, but allow positional argument to be an addable/subtractable object (datetime, timedelta, dateutil.relativedelta). **breaking change**

7.3.22 v0.3.0 (2016-08-03)

- Rename `DateTime.sub` to `DateTime.subtract`. **breaking change**
- Allow the first argument to `DateTime.add` to be a `datetime.timedelta` or `dateutil.relativedelta` object.
- Allow the first argument to `DateTime.subtract` to be a `DateTime`, `datetime.datetime`, `datetime.timedelta`, or `dateutil.relativedelta` object.
- Provide `zulu.ISO8601` and `zulu.TIMESTAMP` as parse/format constants that can be used in `zulu.parse(string, zulu.ISO8601)` and `DateTime.format(zulu.ISO8601)`.
- Remove special parse format string `'timestamp'` in favor of using just `'X'` as defined in `zulu.TIMESTAMP`. **breaking change**
- Import `zulu.parser.format` to `zulu.format`.
- Fix bug in `DateTime` addition operation that resulted in a native `datetime` being returned instead of `DateTime`.

7.3.23 v0.2.0 (2016-08-02)

- Add `DateTime.datetime` property that returns a native datetime.
- Add `DateTime.fromgmtime` that creates a `DateTime` from a UTC based `time.struct_time`.
- Add `DateTime.fromlocaltime` that creates a `DateTime` from a local `time.struct_time`.
- Add `DateTime.isleap` method that returns whether its year is a leap year.
- Add `DateTime.leapdays` that calculates the number of leap days between its year and another year.
- Add `DateTime.start_of/end_of` and other variants that return the start of end of a time frame:
  - `start/end_of_century`
  - `start/end_of_decade`
  - `start/end_of_year`
  - `start/end_of_month`
  - `start/end_of_day`
  - `start/end_of_hour`
  - `start/end_of_minute`
  - `start/end_of_second`
- Add `DateTime.span` that returns the start and end of a time frame.
- Add `DateTime.span_range` that returns a range of spans.
- Add `DateTime.range` that returns a range of datetimes.
- Add `DateTime.add` and `DateTime.sub` methods.
- Add `years` and `months` arguments to `DateTime.shift/add/sub`.
- Drop support for milliseconds from `DateTime.shift/add/sub`. **breaking change**
- Make `DateTime.parse/format` understand a subset of Unicode date patterns.

- Set defaults for year (1970), month (1), and day (1) arguments to new `DateTime` objects. Creating a new `DateTime` now defaults to the start of the POSIX epoch.

7.3.24 v0.1.2 (2016-07-26)

- Don't pin install requirements to a specific version; use >= instead.

7.3.25 v0.1.1 (2016-07-26)

- Fix bug in `DateTime.naive` that resulted in a `DateTime` object being returned instead of a native `datetime`.

7.3.26 v0.1.0 (2016-07-26)

- First release.

7.4 Authors

7.4.1 Lead

- Derrick Gilland, dgilland@gmail.com, dgilland@github

7.4.2 Contributors

- Bharadwaj Yarlagadda, yarlagaddabharadwaj@gmail.com, bharadwajyarlagadda@github
- Thomas Chiroux, ThomasChiroux@github

7.5 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

7.5.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/dgilland/zulu.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" or "help wanted" is open to whoever wants to implement it.

Write Documentation

zulu could always use more documentation, whether as part of the official zulu docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/dgilland/zulu.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

7.5.2 Get Started!

Ready to contribute? Here's how to set up `zulu` for local development.

1. Fork the `zulu` repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_username_here/zulu.git
```

3. Install Python dependencies into a virtualenv:

```
$ cd zulu
$ pip install -r requirements.txt
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. Autoformat code:

```
$ inv fmt
```

6. When you're done making changes, check that your changes pass all unit tests by testing with `tox` across all supported Python versions:

```
$ tox
```

7. Add yourself to `AUTHORS.rst`.
8. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "<Detailed description of your changes>"
$ git push origin name-of-your-bugfix-or-feature-branch
```

9. Submit a pull request through GitHub.

7.5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. The pull request should work for all versions Python that this project supports.

CHAPTER 8

Indices and Tables

- genindex
- modindex
- search

Index

T

U

W

Z