aniso8601 Documentation

Release 1.3.0

Brandon Nielsen

Sep 01, 2017

Contents

CHAPTER 1

Another ISO 8601 parser for Python

Features

- Pure Python implementation
- Python 3 support
- Logical behavior
  - Parse a time, get a datetime.time
  - Parse a date, get a datetime.date
  - Parse a datetime, get a datetime.datetime
  - Parse a duration, get a datetime.timedelta
  - Parse an interval, get a tuple of dates or datetimes
  - Parse a repeating interval, get a date or datetime generator
- UTC offset represented as fixed-offset tzinfo
- dateutil.relativedelta available for calendar accuracy
- No regular expressions

Installation

The recommended installation method is to use pip:

```
$ pip install aniso8601
```

Alternatively, you can download the source (git repository hosted at Bitbucket) and install directly:

```
$ python setup.py install
```

Use

Parsing datetimes

To parse a typical ISO 8601 datetime string:

```
>>> import aniso8601
>>> aniso8601.parse_datetime('1977-06-10T12:00:00Z')
datetime.datetime(1977, 6, 10, 12, 0, tzinfo=+0:00:00 UTC)
```

Alternative delimiters can be specified, for example, a space:

```
>>> aniso8601.parse_datetime('1977-06-10 12:00:00Z', delimiter=' ')
datetime.datetime(1977, 6, 10, 12, 0, tzinfo=+0:00:00 UTC)
```

UTC offsets are supported:

```
>>> aniso8601.parse_datetime('1979-06-05T08:00:00-08:00')
datetime.datetime(1979, 6, 5, 8, 0, tzinfo=-8:00:00 UTC)
```

If a UTC offset is not specified, the returned datetime will be naive:

```
>>> aniso8601.parse_datetime('1983-01-22T08:00:00')
datetime.datetime(1983, 1, 22, 8, 0)
```

Parsing dates

To parse a date represented in an ISO 8601 string:

```
>>> import aniso8601
>>> aniso8601.parse_date('1984-04-23')
datetime.date(1984, 4, 23)
```

Basic format is supported as well:

```
>>> aniso8601.parse_date('19840423')
datetime.date(1984, 4, 23)
```

To parse a date using the ISO 8601 week date format:

```
>>> aniso8601.parse_date('1986-W38-1')
datetime.date(1986, 9, 15)
```

To parse an ISO 8601 ordinal date:

```
>>> aniso8601.parse_date('1988-132')
datetime.date(1988, 5, 11)
```

Parsing times

To parse a time formatted as an ISO 8601 string:

```
>>> import aniso8601
>>> aniso8601.parse_time('11:31:14')
datetime.time(11, 31, 14)
```

As with all of the above, basic format is supported:

```
>>> aniso8601.parse_time('113114')
datetime.time(11, 31, 14)
```

A UTC offset can be specified for times:

```
>>> aniso8601.parse_time('17:18:19-02:30')
datetime.time(17, 18, 19, tzinfo=-2:30:00 UTC)
>>> aniso8601.parse_time('171819Z')
datetime.time(17, 18, 19, tzinfo=+0:00:00 UTC)
```

Reduced accuracy is supported:

```
>>> aniso8601.parse_time('21:42')
datetime.time(21, 42)
>>> aniso8601.parse_time('22')
datetime.time(22, 0)
```

A decimal fraction is always allowed on the lowest order element of an ISO 8601 formatted time:

```
>>> aniso8601.parse_time('22:33.5')
datetime.time(22, 33, 30)
>>> aniso8601.parse_time('23.75')
datetime.time(23, 45)
```

Leap seconds are currently not supported:

```
>>> aniso8601.parse_time('21:42:60')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "aniso8601/time.py", line 107, in parse_time
    return _parse_time_naive(timestr)
  File "aniso8601/time.py", line 136, in _parse_time_naive
    return _resolution_map[get_time_resolution(timestr)](timestr)
  File "aniso8601/time.py", line 199, in _parse_second_time
    raise ValueError('Seconds must be less than 60.')
ValueError: Seconds must be less than 60.
```

Parsing durations

To parse a duration formatted as an ISO 8601 string:

```
>>> import aniso8601
>>> aniso8601.parse_duration('P1Y2M3DT4H54M6S')
datetime.timedelta(428, 17646)
```

Reduced accuracy is supported:

```
>>> aniso8601.parse_duration('P1Y')
datetime.timedelta(365)
```

A decimal fraction is allowed on the lowest order element:

```
>>> aniso8601.parse_duration('P1YT3.5M')
datetime.timedelta(365, 210)
```

The decimal fraction can be specified with a comma instead of a full-stop:

```
>>> aniso8601.parse_duration('P1YT3,5M')
datetime.timedelta(365, 210)
```

Parsing a duration from a combined date and time is supported as well:

```
>>> aniso8601.parse_duration('P0001-01-02T01:30:5')
datetime.timedelta(397, 5405)
```

The above treat years as 365 days and months as 30 days. If calendar level accuracy is required, the relative keyword argument can be used:

```
>>> import aniso8601
>>> from datetime import date
>>> one_month = aniso8601.parse_duration('P1M', relative=True)
>>> print one_month
relativedelta(months=+1)
>>> date(2003,1,27) + one_month
datetime.date(2003, 2, 27)
>>> date(2003,1,31) + one_month
datetime.date(2003, 2, 28)
>>> date(2003,1,31) + two_months
datetime.date(2003, 3, 31)
```

Since a relative fractional month or year is not logical, a ValueError is raised when attempting to parse a duration with `relative=True` and fractional month or year:

```
>>> aniso8601.parse_duration('P2.1Y', relative=True)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "aniso8601/duration.py", line 29, in parse_duration
    return _parse_duration_prescribed(isodurationstr, relative)
  File "aniso8601/duration.py", line 150, in _parse_duration_prescribed
    raise ValueError('Fractional months and years are not defined for relative␣
↪intervals.')
ValueError: Fractional months and years are not defined for relative intervals.
```

Parsing intervals

To parse an interval specified by a start and end:

```
>>> import aniso8601
>>> aniso8601.parse_interval('2007-03-01T13:00:00/2008-05-11T15:30:00')
(datetime.datetime(2007, 3, 1, 13, 0), datetime.datetime(2008, 5, 11, 15, 30))
```

Intervals specified by a start time and a duration are supported:

```
>>> aniso8601.parse_interval('2007-03-01T13:00:00Z/P1Y2M10DT2H30M')
(datetime.datetime(2007, 3, 1, 13, 0, tzinfo=+0:00:00 UTC), datetime.datetime(2008, 5,
↪ 9, 15, 30, tzinfo=+0:00:00 UTC))
```

A duration can also be specified by a duration and end time:

```
>>> aniso8601.parse_interval('P1M/1981-04-05')
(datetime.date(1981, 4, 5), datetime.date(1981, 3, 6))
```

Notice that the result of the above parse is not in order from earliest to latest. If sorted intervals are required, simply use the 'sorted' keyword as shown below:

```
>>> sorted(aniso8601.parse_interval('P1M/1981-04-05'))
[datetime.date(1981, 3, 6), datetime.date(1981, 4, 5)]
```

The end of an interval is given as a datetime when required to maintain the resolution specified by a duration, even if the duration start is given as a date:

```
>>> aniso8601.parse_interval('2014-11-12/PT4H54M6.5S')
(datetime.date(2014, 11, 12), datetime.datetime(2014, 11, 12, 4, 54, 6, 500000))
```

Repeating intervals are supported as well, and return a generator:

```
>>> aniso8601.parse_repeating_interval('R3/1981-04-05/P1D')
<generator object date_generator at 0x7f698cdefc80>
>>> list(aniso8601.parse_repeating_interval('R3/1981-04-05/P1D'))
[datetime.date(1981, 4, 5), datetime.date(1981, 4, 6), datetime.date(1981, 4, 7)]
```

Repeating intervals are allowed to go in the reverse direction:

```
>>> list(aniso8601.parse_repeating_interval('R2/PT1H2M/1980-03-05T01:01:00'))
[datetime.datetime(1980, 3, 5, 1, 1), datetime.datetime(1980, 3, 4, 23, 59)]
```

Unbounded intervals are also allowed (Python 2):

```
>>> result = aniso8601.parse_repeating_interval('R/PT1H2M/1980-03-05T01:01:00')
>>> result.next()
datetime.datetime(1980, 3, 5, 1, 1)
>>> result.next()
datetime.datetime(1980, 3, 4, 23, 59)
```

or for Python 3:

```
>>> result = aniso8601.parse_repeating_interval('R/PT1H2M/1980-03-05T01:01:00')
>>> next(result)
datetime.datetime(1980, 3, 5, 1, 1)
>>> next(result)
datetime.datetime(1980, 3, 4, 23, 59)
```

Note that you should never try to convert a generator produced by an unbounded interval to a list:

```
>>> list(aniso8601.parse_repeating_interval('R/PT1H2M/1980-03-05T01:01:00'))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "aniso8601/interval.py", line 156, in _date_generator_unbounded
    currentdate += timedelta
OverflowError: date value out of range
```

The above treat years as 365 days and months as 30 days. If calendar level accuracy is required, the relative keyword argument can be used:

```
>>> aniso8601.parse_interval('2003-01-27/P1M', relative=True)
(datetime.date(2003, 1, 27), datetime.date(2003, 2, 27))
>>> aniso8601.parse_interval('2003-01-31/P1M', relative=True)
(datetime.date(2003, 1, 31), datetime.date(2003, 2, 28))
>>> aniso8601.parse_interval('P1Y/2001-02-28', relative=True)
(datetime.date(2001, 2, 28), datetime.date(2000, 2, 28)
```

Fractional years and months do not make sense for relative intervals. A ValueError is raised when attempting to parse an interval with `relative=True` and a fractional month or year:

```
>>> aniso8601.parse_interval('P1.1Y/2001-02-28', relative=True)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "aniso8601/interval.py", line 33, in parse_interval
    interval_parts = _parse_interval_parts(isointervalstr, intervaldelimiter,␣
↪datetimedelimiter, relative)
  File "aniso8601/interval.py", line 84, in _parse_interval_parts
    duration = parse_duration(firstpart, relative=relative)
  File "aniso8601/duration.py", line 29, in parse_duration
    return _parse_duration_prescribed(isodurationstr, relative)
  File "aniso8601/duration.py", line 150, in _parse_duration_prescribed
    raise ValueError('Fractional months and years are not defined for relative␣
↪intervals.')
ValueError: Fractional months and years are not defined for relative intervals.
```

Date and time resolution

In some situations, it may be useful to figure out the resolution provided by an ISO 8601 date or time string. Two functions are provided for this purpose.

To get the resolution of a ISO 8601 time string:

```
>>> aniso8601.get_time_resolution('11:31:14') == aniso8601.resolution.TimeResolution.
↪Seconds
True
>>> aniso8601.get_time_resolution('11:31') == aniso8601.resolution.TimeResolution.
↪Minutes
True
>>> aniso8601.get_time_resolution('11') == aniso8601.resolution.TimeResolution.Hours
True
```

Similarly, for an ISO 8601 date string:

```
>>> aniso8601.get_date_resolution('1981-04-05') == aniso8601.resolution.
↪DateResolution.Day
True
>>> aniso8601.get_date_resolution('1981-04') == aniso8601.resolution.DateResolution.
↪Month
True
>>> aniso8601.get_date_resolution('1981') == aniso8601.resolution.DateResolution.Year
True
```

Tests

To run the unit tests, navigate to the source directory and run the tests for the python version being worked on (python2, python3):

```
$ python2 -m unittest discover aniso8601/tests/
```

or:

```
$ python3 -m unittest discover aniso8601/tests/
```

Contributing

aniso8601 is an open source project hosted on Bitbucket.

Any and all bugs are welcome on our issue tracker. Of particular interest are valid ISO 8601 strings that don't parse, or invalid ones that do. At a minimum, bug reports should include an example of the misbehaving string, as well as the expected result. Of course patches containing unit tests (or fixed bugs) are welcome!

References

- ISO 8601:2004(E) (Caution, PDF link)
- Wikipedia article on ISO 8601
- Discussion on alternative ISO 8601 parsers for Python