
aniso8601 Documentation

Release 6.0.0

Brandon Nielsen

Mar 08, 2019

Contents

1	Another ISO 8601 parser for Python	1
2	Features	3
3	Installation	5
4	Use	7
4.1	Parsing datetimes	7
4.2	Parsing dates	8
4.3	Parsing times	8
4.4	Parsing durations	9
4.5	Parsing intervals	10
4.6	Date and time resolution	11
5	Builders	13
5.1	TupleBuilder	13
6	Development	17
6.1	Setup	17
6.2	Tests	17
7	Contributing	19
8	References	21

CHAPTER 1

Another ISO 8601 parser for Python

- 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
- Parser separate from representation, allowing parsing to different datetime formats
- No regular expressions

CHAPTER 3

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
```


4.1 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)
```

Leap seconds are currently not supported and attempting to parse one raises a `LeapSecondError`:

```
>>> aniso8601.parse_datetime('2018-03-06T23:59:60')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "aniso8601/time.py", line 131, in parse_datetime
    return builder.build_datetime(datepart, timepart)
  File "aniso8601/builder.py", line 300, in build_datetime
    cls._build_object(time)
  File "aniso8601/builder.py", line 71, in _build_object
```

(continues on next page)

(continued from previous page)

```
ss=parsetuple[2], tz=parsetuple[3])
File "aniso8601/builder.py", line 253, in build_time
    raise LeapSecondError('Leap seconds are not supported.')
aniso8601.exceptions.LeapSecondError: Leap seconds are not supported.
```

4.2 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)
```

4.3 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 and attempting to parse one raises a LeapSecondError:

```
>>> aniso8601.parse_time('23:59:60')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "aniso8601/time.py", line 116, in parse_time
    return _RESOLUTION_MAP[get_time_resolution(timestr)](timestr, tz, builder)
  File "aniso8601/time.py", line 165, in _parse_second_time
    return builder.build_time(hh=hourstr, mm=minuteistr, ss=secondstr, tz=tz)
  File "aniso8601/builder.py", line 253, in build_time
    raise LeapSecondError('Leap seconds are not supported.')
aniso8601.exceptions.LeapSecondError: Leap seconds are not supported.
```

4.4 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)
```

4.5 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 returned 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))
>>> aniso8601.parse_interval('2007-03-01/P1.5D')
(datetime.date(2007, 3, 1), datetime.datetime(2007, 3, 2, 12, 0))
```

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

```
>>> aniso8601.parse_repeating_interval('R3/1981-04-05/P1D')
<generator object _date_generator at 0x7fd800d3b320>
>>> 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/builders/python.py", line 463, in _date_generator_unbounded
    currentdate += timedelta
OverflowError: date value out of range
```

4.6 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
```


Builders can be used to change the output format of a parse operation. All parse functions have a `builder` keyword argument which accepts a builder class.

Two builders are included. The `PythonTimeBuilder` (the default) in the `aniso8601.builders.python` module, and the `TupleBuilder` which returns the parse result as a tuple of strings and is located in the `aniso8601.builders` module.

The following builders are available as separate projects:

- `RelativeTimeBuilder` supports parsing to **'dateutil relativedelta types'** <[https://dateutil.readthedocs.io/en/stable/relativedelta.h](https://dateutil.readthedocs.io/en/stable/relativedelta.html)> for calendar level accuracy
- `AttoTimeBuilder` <<https://bitbucket.org/nielsenb/attotimebuilder>>_ supports parsing directly to `attotime` `attodatetime` and `attotimedelta` types which support sub-nanosecond precision
- `NumPyTimeBuilder` supports parsing directly to `NumPy` `datetime64` and `timedelta64` types

5.1 TupleBuilder

The `TupleBuilder` returns parse results as tuples of strings. It is located in the `aniso8601.builders` module.

5.1.1 Datetimes

Parsing a datetime returns a tuple containing a date tuple as a collection of strings, a time tuple as a collection of strings, and the 'datetime' string. The date tuple contains the following parse components: (YYYY, MM, DD, Www, D, DDD, 'date'). The time tuple contains the following parse components (hh, mm, ss, tz, 'time'), where tz is a tuple with the following components (negative, Z, hh, mm, name, 'timezone') with negative and Z being booleans:

```
>>> import aniso8601
>>> from aniso8601.builders import TupleBuilder
>>> aniso8601.parse_datetime('1977-06-10T12:00:00', builder=TupleBuilder)
```

(continues on next page)

(continued from previous page)

```
((('1977', '06', '10', None, None, None, 'date'), ('12', '00', '00', None, 'time'),  
→ 'datetime'))  
>>> aniso8601.parse_datetime('1979-06-05T08:00:00-08:00', builder=TupleBuilder)  
((('1979', '06', '05', None, None, None, 'date'), ('08', '00', '00', (True, None, '08',  
→ '00', '-08:00', 'timezone'), 'time'), 'datetime'))
```

5.1.2 Dates

Parsing a date returns a tuple containing the following parse components: (YYYY, MM, DD, Www, D, DDD, 'date'):

```
>>> import aniso8601  
>>> from aniso8601.builders import TupleBuilder  
>>> aniso8601.parse_date('1984-04-23', builder=TupleBuilder)  
('1984', '04', '23', None, None, None, 'date')  
>>> aniso8601.parse_date('1986-W38-1', builder=TupleBuilder)  
('1986', None, None, '38', '1', None, 'date')  
>>> aniso8601.parse_date('1988-132', builder=TupleBuilder)  
('1988', None, None, None, None, '132', 'date')
```

5.1.3 Times

Parsing a time returns a tuple containing following parse components: (hh, mm, ss, tz, 'time'), where tz is a tuple with the following components (negative, Z, hh, mm, name, 'timezone') with negative and Z being booleans:

```
>>> import aniso8601  
>>> from aniso8601.builders import TupleBuilder  
>>> aniso8601.parse_time('11:31:14', builder=TupleBuilder)  
('11', '31', '14', None, 'time')  
>>> aniso8601.parse_time('171819Z', builder=TupleBuilder)  
('17', '18', '19', (False, True, None, None, 'Z', 'timezone'), 'time')  
>>> aniso8601.parse_time('17:18:19-02:30', builder=TupleBuilder)  
('17', '18', '19', (True, None, '02', '30', '-02:30', 'timezone'), 'time')
```

5.1.4 Durations

Parsing a duration returns a tuple containing the following parse components: (PnY, PnM, PnW, PnD, TnH, TnM, TnS, 'duration'):

```
>>> import aniso8601  
>>> from aniso8601.builders import TupleBuilder  
>>> aniso8601.parse_duration('P1Y2M3DT4H54M6S', builder=TupleBuilder)  
('1', '2', None, '3', '4', '54', '6', 'duration')  
>>> aniso8601.parse_duration('P7W', builder=TupleBuilder)  
(None, None, '7', None, None, None, None, 'duration')
```

5.1.5 Intervals

Parsing an interval returns a tuple containing the following parse components: (start, end, duration, 'interval'), start and end may both be datetime or date tuples, duration is a duration tuple:

```
>>> import aniso8601
>>> from aniso8601.builders import TupleBuilder
>>> aniso8601.parse_interval('2007-03-01T13:00:00/2008-05-11T15:30:00',
↳builder=TupleBuilder)
(((('2007', '03', '01', None, None, None, 'date'), ('13', '00', '00', None, 'time'),
↳'datetime'), (('2008', '05', '11', None, None, None, 'date'), ('15', '30', '00',
↳None, 'time'), 'datetime'), None, 'interval')
>>> aniso8601.parse_interval('2007-03-01T13:00:00Z/P1Y2M10DT2H30M',
↳builder=TupleBuilder)
(((('2007', '03', '01', None, None, None, 'date'), ('13', '00', '00', (False, True,
↳None, None, 'Z', 'timezone'), 'time'), 'datetime'), None, ('1', '2', None, '10', '2
↳', '30', None, 'duration'), 'interval')
>>> aniso8601.parse_interval('P1M/1981-04-05', builder=TupleBuilder)
(None, ('1981', '04', '05', None, None, None, 'date'), (None, '1', None, None, None,
↳None, None, 'duration'), 'interval')
```

A repeating interval returns a tuple containing the following parse components: (R, Rnn, interval, 'repeatinginterval') where R is a boolean, True for an unbounded interval, False otherwise.:

```
>>> aniso8601.parse_repeating_interval('R3/1981-04-05/P1D', builder=TupleBuilder)
(False, '3', (('1981', '04', '05', None, None, None, 'date'), None, (None, None, None,
↳'1', None, None, None, 'duration'), 'interval'), 'repeatinginterval')
>>> aniso8601.parse_repeating_interval('R/PT1H2M/1980-03-05T01:01:00',
↳builder=TupleBuilder)
(True, None, (None, (('1980', '03', '05', None, None, None, 'date'), ('01', '01', '00
↳', None, 'time'), 'datetime'), (None, None, None, None, '1', '2', None, 'duration'),
↳'interval'), 'repeatinginterval')
```


6.1 Setup

It is recommended to develop using a `virtualenv`.

6.2 Tests

Tests can be run using `setuptools` <<https://setuptools.readthedocs.io/en/latest/setuptools.html>>:

```
$ python setup.py test
```


CHAPTER 7

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!

CHAPTER 8

References

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