aniso8601

Release 9.0.0

Feb 18, 2021

Contents

CHAPTER 1

Another ISO 8601 parser for Python

CHAPTER 2

Features

- Pure Python implementation
- 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 representations (see Builders)
- 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
```

CHAPTER 4

Use

4.1 Parsing datetimes

Consider datetime.datetime.fromisoformat *for basic ISO 8601 datetime parsing*

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 "/home/nielsenb/Jetfuse/aniso8601/aniso8601/aniso8601/time.py", line 196, in␣
↪parse_datetime
    return builder.build_datetime(datepart, timepart)
```

(continues on next page)

(continued from previous page)

```
  File "/home/nielsenb/Jetfuse/aniso8601/aniso8601/aniso8601/builders/python.py",␣
↪line 237, in build_datetime
    cls._build_object(time))
  File "/home/nielsenb/Jetfuse/aniso8601/aniso8601/aniso8601/builders/__init__.py",␣
↪line 336, in _build_object
    return cls.build_time(hh=parsetuple.hh, mm=parsetuple.mm,
  File "/home/nielsenb/Jetfuse/aniso8601/aniso8601/aniso8601/builders/python.py",␣
↪line 191, in build_time
    hh, mm, ss, tz = cls.range_check_time(hh, mm, ss, tz)
  File "/home/nielsenb/Jetfuse/aniso8601/aniso8601/aniso8601/builders/__init__.py",␣
↪line 266, in range_check_time
    raise LeapSecondError('Leap seconds are not supported.')
aniso8601.exceptions.LeapSecondError: Leap seconds are not supported.
```

To get the resolution of an ISO 8601 datetime string:

```
>>> aniso8601.get_datetime_resolution('1977-06-10T12:00:00Z') == aniso8601.resolution.
↪TimeResolution.Seconds
True
>>> aniso8601.get_datetime_resolution('1977-06-10T12:00') == aniso8601.resolution.
↪TimeResolution.Minutes
True
>>> aniso8601.get_datetime_resolution('1977-06-10T12') == aniso8601.resolution.
↪TimeResolution.Hours
True
```

Note that datetime resolutions map to `TimeResolution` as a valid datetime must have at least one time member so the resolution mapping is equivalent.

4.2 Parsing dates

Consider datetime.date.fromisoformat *for basic ISO 8601 date parsing*

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)
```

To get the resolution of 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
```

4.3 Parsing times

Consider datetime.time.fromisoformat *for basic ISO 8601 time parsing*

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)
```

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

```
>>> 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 "/home/nielsenb/Jetfuse/aniso8601/aniso8601/aniso8601/time.py", line 174, in␣
↪parse_time
    return builder.build_time(hh=hourstr, mm=minutestr, ss=secondstr, tz=tz)
  File "/home/nielsenb/Jetfuse/aniso8601/aniso8601/aniso8601/builders/python.py",␣
↪line 191, in build_time
    hh, mm, ss, tz = cls.range_check_time(hh, mm, ss, tz)
  File "/home/nielsenb/Jetfuse/aniso8601/aniso8601/aniso8601/builders/__init__.py",␣
↪line 266, in range_check_time
    raise LeapSecondError('Leap seconds are not supported.')
aniso8601.exceptions.LeapSecondError: Leap seconds are not supported.
```

To get the resolution of an 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
```

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('P0001-01-02T01:30:05')
datetime.timedelta(397, 5405)
```

To get the resolution of an ISO 8601 duration string:

```
>>> aniso8601.get_duration_resolution('P1Y2M3DT4H54M6S') == aniso8601.resolution.
↪DurationResolution.Seconds
True
>>> aniso8601.get_duration_resolution('P1Y2M3DT4H54M') == aniso8601.resolution.
↪DurationResolution.Minutes
True
>>> aniso8601.get_duration_resolution('P1Y2M3DT4H') == aniso8601.resolution.
↪DurationResolution.Hours
True
>>> aniso8601.get_duration_resolution('P1Y2M3D') == aniso8601.resolution.
↪DurationResolution.Days
True
>>> aniso8601.get_duration_resolution('P1Y2M') == aniso8601.resolution.
↪DurationResolution.Months
True
>>> aniso8601.get_duration_resolution('P1Y') == aniso8601.resolution.
↪DurationResolution.Years
True
```

The default `PythonTimeBuilder` assumes years are 365 days, and months are 30 days. Where calendar level accuracy is required, a RelativeTimeBuilder can be used, see also Builders.

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))
```

Concise representations are supported:

```
>>> aniso8601.parse_interval('2020-01-01/02')
(datetime.date(2020, 1, 1), datetime.date(2020, 1, 2))
>>> aniso8601.parse_interval('2007-12-14T13:30/15:30')
(datetime.datetime(2007, 12, 14, 13, 30), datetime.datetime(2007, 12, 14, 15, 30))
>>> aniso8601.parse_interval('2008-02-15/03-14')
(datetime.date(2008, 2, 15), datetime.date(2008, 3, 14))
>>> aniso8601.parse_interval('2007-11-13T09:00/15T17:00')
(datetime.datetime(2007, 11, 13, 9, 0), datetime.datetime(2007, 11, 15, 17, 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 "/home/nielsenb/Jetfuse/aniso8601/aniso8601/aniso8601/builders/python.py",␣
↪line 560, in _date_generator_unbounded
    currentdate += timedelta
OverflowError: date value out of range
```

To get the resolution of an ISO 8601 interval string:

```
>>> aniso8601.get_interval_resolution('2007-03-01T13:00:00/2008-05-11T15:30:00') ==␣
↪aniso8601.resolution.IntervalResolution.Seconds
True
>>> aniso8601.get_interval_resolution('2007-03-01T13:00/2008-05-11T15:30') ==␣
↪aniso8601.resolution.IntervalResolution.Minutes
True
>>> aniso8601.get_interval_resolution('2007-03-01T13/2008-05-11T15') == aniso8601.
↪resolution.IntervalResolution.Hours
```

(continues on next page)

(continued from previous page)

```
True
>>> aniso8601.get_interval_resolution('2007-03-01/2008-05-11') == aniso8601.
↪resolution.IntervalResolution.Day
True
>>> aniso8601.get_interval_resolution('2007-03/P1Y') == aniso8601.resolution.
↪IntervalResolution.Month
True
>>> aniso8601.get_interval_resolution('2007/P1Y') == aniso8601.resolution.
↪IntervalResolution.Year
True
```

And for repeating ISO 8601 interval strings:

```
>>> aniso8601.get_repeating_interval_resolution('R3/1981-04-05/P1D') == aniso8601.
↪resolution.IntervalResolution.Day
True
>>> aniso8601.get_repeating_interval_resolution('R/PT1H2M/1980-03-05T01:01:00') ==␣
↪aniso8601.resolution.IntervalResolution.Seconds
True
```

CHAPTER 5

Builders

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 corresponding named tuple and is located in the `aniso8601.builders` module.

Information on writing a builder can be found in BUILDERS.

The following builders are available as separate projects:

- RelativeTimeBuilder supports parsing to datetutil relativedelta types for calendar level accuracy
- 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 named tuples. It is located in the `aniso8601.builders` module.

5.1.1 Datetimes

Parsing a datetime returns a `DatetimeTuple` containing `Date` and `Time` tuples . The date tuple contains the following parse components: `YYYY`, `MM`, `DD`, `Www`, `D`, `DDD`. The time tuple contains the following parse components `hh`, `mm`, `ss`, `tz`, where `tz` itself is a tuple with the following components `negative`, `Z`, `hh`, `mm`, `name` 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)

```
Datetime(date=Date(YYYY='1977', MM='06', DD='10', Www=None, D=None, DDD=None),␣
↪time=Time(hh='12', mm='00', ss='00', tz=None))
>>> aniso8601.parse_datetime('1979-06-05T08:00:00-08:00', builder=TupleBuilder)
Datetime(date=Date(YYYY='1979', MM='06', DD='05', Www=None, D=None, DDD=None),␣
↪time=Time(hh='08', mm='00', ss='00', tz=Timezone(negative=True, Z=None, hh='08', mm=
↪'00', name='-08:00')))
```

5.1.2 Dates

Parsing a date returns a `DateTuple` containing the following parse components: `YYYY`, `MM`, `DD`, `Www`, `D`, `DDD`:

```
>>> import aniso8601
>>> from aniso8601.builders import TupleBuilder
>>> aniso8601.parse_date('1984-04-23', builder=TupleBuilder)
Date(YYYY='1984', MM='04', DD='23', Www=None, D=None, DDD=None)
>>> aniso8601.parse_date('1986-W38-1', builder=TupleBuilder)
Date(YYYY='1986', MM=None, DD=None, Www='38', D='1', DDD=None)
>>> aniso8601.parse_date('1988-132', builder=TupleBuilder)
Date(YYYY='1988', MM=None, DD=None, Www=None, D=None, DDD='132')
```

5.1.3 Times

Parsing a time returns a `TimeTuple` containing following parse components: `hh`, `mm`, `ss`, `tz`, where `tz` is a `TimezoneTuple` with the following components `negative`, `Z`, `hh`, `mm`, `name`, with `negative` and `Z` being booleans:

```
>>> import aniso8601
>>> from aniso8601.builders import TupleBuilder
>>> aniso8601.parse_time('11:31:14', builder=TupleBuilder)
Time(hh='11', mm='31', ss='14', tz=None)
>>> aniso8601.parse_time('171819Z', builder=TupleBuilder)
Time(hh='17', mm='18', ss='19', tz=Timezone(negative=False, Z=True, hh=None, mm=None,␣
↪name='Z'))
>>> aniso8601.parse_time('17:18:19-02:30', builder=TupleBuilder)
Time(hh='17', mm='18', ss='19', tz=Timezone(negative=True, Z=None, hh='02', mm='30',␣
↪name='-02:30'))
```

5.1.4 Durations

Parsing a duration returns a `DurationTuple` containing the following parse components: `PnY`, `PnM`, `PnW`, `PnD`, `TnH`, `TnM`, `TnS`:

```
>>> import aniso8601
>>> from aniso8601.builders import TupleBuilder
>>> aniso8601.parse_duration('P1Y2M3DT4H54M6S', builder=TupleBuilder)
Duration(PnY='1', PnM='2', PnW=None, PnD='3', TnH='4', TnM='54', TnS='6')
>>> aniso8601.parse_duration('P7W', builder=TupleBuilder)
Duration(PnY=None, PnM=None, PnW='7', PnD=None, TnH=None, TnM=None, TnS=None)
```

5.1.5 Intervals

Parsing an interval returns an `IntervalTuple` containing the following parse components: `start`, `end`, `duration`, `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)
Interval(start=Datetime(date=Date(YYYY='2007', MM='03', DD='01', Www=None, D=None,␣
↪DDD=None), time=Time(hh='13', mm='00', ss='00', tz=None)),␣
↪end=Datetime(date=Date(YYYY='2008', MM='05', DD='11', Www=None, D=None, DDD=None),␣
↪time=Time(hh='15', mm='30', ss='00', tz=None)), duration=None)
>>> aniso8601.parse_interval('2007-03-01T13:00:00Z/P1Y2M10DT2H30M',␣
↪builder=TupleBuilder)
Interval(start=Datetime(date=Date(YYYY='2007', MM='03', DD='01', Www=None, D=None,␣
↪DDD=None), time=Time(hh='13', mm='00', ss='00', tz=Timezone(negative=False, Z=True,␣
↪hh=None, mm=None, name='Z'))), end=None, duration=Duration(PnY='1', PnM='2',␣
↪PnW=None, PnD='10', TnH='2', TnM='30', TnS=None))
>>> aniso8601.parse_interval('P1M/1981-04-05', builder=TupleBuilder)
Interval(start=None, end=Date(YYYY='1981', MM='04', DD='05', Www=None, D=None,␣
↪DDD=None), duration=Duration(PnY=None, PnM='1', PnW=None, PnD=None, TnH=None,␣
↪TnM=None, TnS=None))
```

A repeating interval returns a `RepeatingIntervalTuple` containing the following parse components: `R`, `Rnn`, `interval`, where `R` is a boolean, `True` for an unbounded interval, `False` otherwise.:

```
>>> aniso8601.parse_repeating_interval('R3/1981-04-05/P1D', builder=TupleBuilder)
RepeatingInterval(R=False, Rnn='3', interval=Interval(start=Date(YYYY='1981', MM='04',
↪ DD='05', Www=None, D=None, DDD=None), end=None, duration=Duration(PnY=None,␣
↪PnM=None, PnW=None, PnD='1', TnH=None, TnM=None, TnS=None)))
>>> aniso8601.parse_repeating_interval('R/PT1H2M/1980-03-05T01:01:00',␣
↪builder=TupleBuilder)
RepeatingInterval(R=True, Rnn=None, interval=Interval(start=None,␣
↪end=Datetime(date=Date(YYYY='1980', MM='03', DD='05', Www=None, D=None, DDD=None),␣
↪time=Time(hh='01', mm='01', ss='00', tz=None)), duration=Duration(PnY=None,␣
↪PnM=None, PnW=None, PnD=None, TnH='1', TnM='2', TnS=None)))
```

CHAPTER 6

Development

6.1 Setup

It is recommended to develop using a virtualenv.

6.2 Tests

Tests can be run using the unittest testing framework:

```
$ python -m unittest discover aniso8601
```

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