

---

# **attotime**

***Release 0.2.1***

**Feb 09, 2021**



---

# Contents

---

<b>1</b>	<b>High precision datetime implementation for Python</b>	<b>1</b>
<b>2</b>	<b>Features</b>	<b>3</b>
<b>3</b>	<b>attotimedelta</b>	<b>5</b>
3.1	<i>class</i> attotime.attotimedelta([days[, seconds[, microseconds[, milliseconds[, minutes[, hours[, weeks[, nanoseconds]]]]]]])) . . . . .	5
3.2	Instance attributes (read-only) . . . . .	5
3.3	Supported operations . . . . .	5
3.4	Instance methods . . . . .	6
<b>4</b>	<b>attodatetime</b>	<b>7</b>
4.1	<i>class</i> attotime.attodatetime(year, month, day[, hour[, minute[, second[, microsecond[, nanosecond[, tzinfo]]]]])) . . . . .	7
4.2	Class methods . . . . .	7
4.3	Instance attributes (read-only) . . . . .	8
4.4	Supported operations . . . . .	9
4.5	Instance methods . . . . .	9
<b>5</b>	<b>attotime</b>	<b>13</b>
5.1	<i>class</i> attotime.attotime([hour[, minute[, second[, microsecond[, nanosecond[, tzinfo]]]]])) . . . . .	13
5.2	Instance attributes (read-only) . . . . .	13
5.3	Supported operations . . . . .	13
5.4	Instance methods . . . . .	14
<b>6</b>	<b>Development</b>	<b>15</b>
6.1	Setup . . . . .	15
6.2	Tests . . . . .	15
<b>7</b>	<b>Contributing</b>	<b>17</b>
<b>8</b>	<b>References</b>	<b>19</b>



# CHAPTER 1

---

## High precision datetime implementation for Python

---



## CHAPTER 2

---

### Features

---

- Fractional nanosecond resolution using Python `decimal` module
- API as close to Python's native `datetime` implementation as possible
- Python 3 support





`attotimedelta` objects represent the difference between two dates or times. It wraps a native `timedelta` object, and stores fractional nanoseconds as a `Decimal`.

### 3.1 `class attotime.attotimedelta([days[, seconds[, microseconds[, milliseconds[, minutes[, hours[, weeks[, nanoseconds]]]]]]])`

All arguments are optional and default to 0. All arguments may be ints, longs, or floats, and may be positive or negative.

Only *days*, *seconds*, *microseconds*, and *nanoseconds* are stored internally. *days*, *seconds*, and *microseconds* are stored in a native `timedelta` object, *nanoseconds* are contained in a `Decimal`.

### 3.2 Instance attributes (read-only)

- `days` Between -999999999 and 999999999 inclusive.
- `seconds` Between 0 and 86399 inclusive.
- `microseconds` Between 0 and 999999 inclusive.
- `nanoseconds` A `Decimal` between 0 and 999 inclusive.

### 3.3 Supported operations

- `td1 = td2 + td3` Sum of `td2` and `td3`.
- `td1 = td2 - td3` Difference of `td2` and `td3`.
- `td1 = i * td2` or `td1 = td2 * i` Delta multiplied by an integer, long, float, or `Decimal`.

- `td1 = td2 // i` Computes the floor, discarding the remainder.
- `+td1` Returns an `attotimedelta` with the same value.
- `-td1` Equivalent to `td1 * -1`.
- `abs(td1)` Equivalent to `+td1` when `td1.days >= 0`, `-td1` when `td1.days < 0`.
- `str(td1)` Returns a string in the form `[D day[s], ][H]H:MM:SS[.UUUUUU]`, where `D` is negative for `td1 < 0` and `UUUUUU` can be expanded for up to 16 place fixed point.
- `repr(td1)` Returns a string in the form `attotime.objects.attotimedelta(D[, S[, U]])`, where `D` is negative for `td1 < 0`.

## 3.4 Instance methods

### 3.4.1 `attotimedelta.total_seconds()`

Return the total number of seconds contained in the duration as a `Decimal`.

### 3.4.2 `attotimedelta.total_nanoseconds()`

Return the total number of nanoseconds contained in the duration as a `Decimal`.

`attodatetime` is a single object wrapping a native `date` object and an `attotime` object for the purposes of storing date and time information with fractional nanoseconds stored as a `Decimal`.

### 4.1 *class* `attotime.attodatetime(year, month, day[, hour[, minute[, second[, microsecond[, nanosecond[, tzinfo]]]]])`

Year, month, and day are required. `tzinfo` may be `None`, or an instance of a `tzinfo` subclass. The nanosecond argument may be a float or `Decimal`. The remaining arguments may be ints or longs.

## 4.2 Class methods

### 4.2.1 `attodatetime.today()`

Return the current local datetime, with `tzinfo` `None`. This is equivalent to `attodatetime.fromtimestamp(time.time())`.

### 4.2.2 `attodatetime.now([tz])`

Return the current local date and time. If optional argument `tz` is `None` this is like `today()`.

If `tz` is not `None`, it must be an instance of a `tzinfo` subclass, and the current date and time are converted to `tz`'s time zone.

### 4.2.3 `attodatetime.utcnow()`

Return the current UTC date and time, with `tzinfo` `None`.

#### 4.2.4 `attodatetime.fromtimestamp(timestamp, [tz])`

Return the local date and time corresponding to the POSIX timestamp, such as returned by `time.time()`. If optional argument `tz` is `None` or not specified, the timestamp is converted to the platform's local date and time, and the returned `attodatetime` object is naive.

If `tz` is not `None`, it must be an instance of a `tzinfo` subclass, and the timestamp is converted to `tz`'s time zone. The returned `attodatetime`'s `tzinfo` is set to the provided `tz`.

#### 4.2.5 `attodatetime.utctimestamp(timestamp)`

Return the UTC `attodatetime` corresponding to the POSIX timestamp, with `tzinfo` `None`.

#### 4.2.6 `attodatetime.fromordinal(ordinal)`

Return an `attodatetime` corresponding to the proleptic Gregorian ordinal, where January 1 of year 1 has ordinal 1. **ValueError** is raised unless  $1 \leq \text{ordinal} \leq \text{datetime.max.toordinal}()$  (note native Python `datetime` range checking). The hour, minute, second and microsecond of the result are all 0, and `tzinfo` is `None`.

#### 4.2.7 `attodatetime.combine(date, time)`

Return an `attodatetime` object whose date components are equal to the given date object's, and whose time components and `tzinfo` attributes are equal to the given time object's. If `date` is a `attodatetime` (or native Python `datetime`), its time components and `tzinfo` attributes are ignored.

#### 4.2.8 `attodatetime.strptime(date_string, format)`

Return an `attodatetime` corresponding to `date_string`, parsed according to `format`. Only the directives explicitly listed in the [`strptime\(\)` and `strftime\(\)` Behavior](#) section of the Python documentation are supported, as well as the following:

Directive	Meaning	Example
<code>%o</code>	Picosecond as a decimal number, zero-padded on the left.	000000, 000001, ..., 999999
<code>%q</code>	Attosecond as a decimal number, zero-padded on the left.	000000, 000001, ..., 999999
<code>%v</code>	Yoctosecond as a decimal number, zero-padded on the left.	000000, 000001, ..., 999999

### 4.3 Instance attributes (read-only)

- `year` Between Python native `datetime` `MINYEAR` and `MAXYEAR`, inclusive.
- `month` Between 1 and 12 inclusive.
- `day` Between 1 and the number of days in the given month of the given year.
- `hour` In range(24).
- `minute` In range(60).
- `second` In range(60).
- `microsecond` In range(1000000).

- `nanosecond` In range (1000), as Decimal.
- `tzinfo` The object passed as the `tzinfo` argument to the `attodatetime` constructor, or `None` if none was passed.

## 4.4 Supported operations

- `dt2 = dt1 + td` `dt1` moved forward the duration of the `attotimedelta` if `attotimedelta.days > 0`, or backward if `attotimedelta.days < 0`.
- `dt2 = dt1 - td` `dt1` moved backward the duration of the `attotimedelta` if `attotimedelta.days > 0`, or forward if `attotimedelta.days < 0`.
- `td = dt1 - dt2` The duration of time between `dt1` and `dt2`, as an `attotimedelta`.
- `dt1 < dt2` `dt1` is considered less than `dt2` if `dt1` precedes `dt2` in time.
- `str(dt1)` Equivalent to `dt1.isoformat(separator=' ')`.
- `repr(dt1)` Returns a string in the form `attotime.objects.attodatetime(Y, M, D, h, m, s, us, ns, [tz])`.

## 4.5 Instance methods

### 4.5.1 `attodatetime.date()`

Return a `date` object with same year, month and day.

### 4.5.2 `attodatetime.time()`

Return an `attotime` object with the same hour, minute, second, microsecond, and nanosecond. `tzinfo` is `None`.

### 4.5.3 `attodatetime.timetz()`

Return an `attotime` object with the same hour, minute, second, microsecond, nanosecond, and `tzinfo` attributes.

### 4.5.4 `attodatetime.replace([year[, month[, day[, hour[, minute[, second[, microsecond[, nanosecond[, tzinfo]]]]]]]])`

Return an `attodatetime` object with the same attributes, except for those attributes given new values by whichever keyword arguments are specified. Note that `tzinfo=None` can be specified to create a naive `attodatetime` from an aware `attodatetime` with no conversion of date and time data.

### 4.5.5 `attodatetime.astimezone(tz)`

Return an `attodatetime` object with new `tzinfo` attribute `tz`, adjusting the date and time data so the result is the same UTC time as self, but in `tz`'s local time.

A `ValueError` is raised if `self` is naive.

#### 4.5.6 attodatetime.utcoffset()

If `tzinfo` is `None`, returns `None`, else return `self.tzinfo.utcoffset(self)` as an `attotimedelta`.

#### 4.5.7 attodatetime.dst()

If `tzinfo` is `None`, returns `None`, else return `self.tzinfo.dst(self)` as an `attotimedelta`.

#### 4.5.8 attodatetime.tzname()

If `tzinfo` is `None`, returns `None`, else returns `self.tzinfo.tzname(self)`.

#### 4.5.9 attodatetime.timetuple()

Return the result of `datetime.timetuple()` for a native Python `datetime` matching the `attodatetime`. Nanosecond precision is lost.

#### 4.5.10 attodatetime.utctimetuple()

Return the result of `datetime.utctimetuple()` for a native Python `datetime` matching the `attodatetime`. Nanosecond precision is lost.

#### 4.5.11 attodatetime.toordinal()

Return the proleptic Gregorian ordinal of the date. The same as `self.date().toordinal()`.

#### 4.5.12 attodatetime.weekday()

Return the day of the week as an integer, where Monday is 0 and Sunday is 6. The same as `self.date().weekday()`.

#### 4.5.13 attodatetime.isoweekday()

Return the day of the week as an integer, where Monday is 1 and Sunday is 7. The same as `self.date().isoweekday()`.

#### 4.5.14 attodatetime.isocalendar()

Return a 3-tuple, (ISO year, ISO week number, ISO weekday). The same as `self.date().isocalendar()`.

#### 4.5.15 attodateime.isoformat([sep])

Return a string representing the date and time in ISO 8601 format, `YYYY-MM-DDTHH:MM:SS.mmmmmmm` or, if microsecond is 0, `YYYY-MM-DDTHH:MM:SS`

If `utcoffset()` does not return `None`, a 6-character string is appended, giving the UTC offset in (signed) hours and minutes: `YYYY-MM-DDTHH:MM:SS.mmmmmmm+HH:MM` or, if microsecond is 0 `YYYY-MM-DDTHH:MM:SS+HH:MM`

The optional argument `sep` (default ‘T’) is a separator, placed between the date and time portions of the result.

The decimal second component may be expanded up to 16 place fixed point.

#### 4.5.16 attodateime.ctime()

Return the result of `datetime.ctime()` for a native Python `datetime` matching the `attodateime`. Nanosecond precision is lost.

#### 4.5.17 attodateime.strftime(format)

Return a string representing the date and time, controlled by an explicit format string. Only the directives explicitly listed in the `strftime()` and `strptime()` Behavior section of the Python documentation are supported, as well as the following:

Directive	Meaning	Example
<code>%o</code>	Picosecond as a decimal number, zero-padded on the left.	000000, 000001, ..., 999999
<code>%q</code>	Attosecond as a decimal number, zero-padded on the left.	000000, 000001, ..., 999999
<code>%v</code>	Yoctosecond as a decimal number, zero-padded on the left.	000000, 000001, ..., 999999





attotime is an object wrapping a native `time` object along with fractional nanoseconds stored as a `Decimal`.

### 5.1 `class attotime.attotime([hour[, minute[, second[, microsecond[, nanosecond[, tzinfo]]]]])`

All arguments are optional. `tzinfo` may be `None`, or an instance of a `tzinfo` subclass. The `nanosecond` argument may be float or `Decimal`. The remaining arguments may be ints or longs.

### 5.2 Instance attributes (read-only)

- `hour` In range(24).
- `minute` In range(60).
- `second` In range(60).
- `microsecond` In range(1000000).
- `nanosecond` In range(1000), as `Decimal`.
- `tzinfo` The object passed as the `tzinfo` argument to the `attotime` constructor, or `None` if none was passed.

### 5.3 Supported operations

- `t1 < t2` `t1` is considered less than `t2` if `t1` precedes `t2` in time.
- `str(t1)` Equivalent to `t1.isoformat()`.

- `repr(t1)` Returns a string in the form `attotime.objects.attotime(h, m, s, us, ns, [tz])`.

## 5.4 Instance methods

### 5.4.1 `attotime.replace([hour[, minute[, second[, microsecond[, nanosecond[, tzinfo]]]])`

Return an `attotime` object with the same attributes, except for those attributes given new values by whichever keyword arguments are specified. Note that `tzinfo=None` can be specified to create a naive `attotime` from an aware `attotime` with no conversion of date and time data.

### 5.4.2 `attotime.isoformat()`

Return a string representing the time in ISO 8601 format, `HH:MM:SS.mmmmmm` or, if `microsecond` is 0, `HH:MM:SS`

If `utcoffset()` does not return `None`, a 6-character string is appended, giving the UTC offset in (signed) hours and minutes: `HH:MM:SS.mmmmmm+HH:MM` or, if `microsecond` is 0 `HH:MM:SS+HH:MM`

The decimal second component may be expanded up to 16 place fixed point.

### 5.4.3 `attotime.strptime(formatstr)`

Raises `NotImplementedError`

### 5.4.4 `attotime.utcoffset()`

If `tzinfo` is `None`, returns `None`, else return `self.tzinfo.utcoffset(self)` as an `attotimedelta`.

### 5.4.5 `attotime.dst()`

If `tzinfo` is `None`, returns `None`, else return `self.tzinfo.dst(self)` as an `attotimedelta`.

### 5.4.6 `attotime.tzname()`

If `tzinfo` is `None`, returns `None`, else returns `self.tzinfo.tzname(self)`.

### 6.1 Setup

It is recommended to develop using a [virtualenv](#).

The tests require the `dev` feature to be enabled, install the necessary dependencies using `pip`:

```
$ pip install .[dev]
```

### 6.2 Tests

Tests can be run using the [unittest testing framework](#):

```
$ python -m unittest discover attotime
```



## CHAPTER 7

---

### Contributing

---

attotime is an open source project hosted on [Bitbucket](#).

Any and all bugs are welcome on our [issue tracker](#). Of particular interest are places where the attotime implementation incorrectly deviates from native Python behavior. Pull requests containing unit tests or fixed bugs are always welcome!



## CHAPTER 8

---

### References

---

- [PEP 410](#) which describes the need for high precision time types
- [Bug report](#) with implementation of PEP 410
- [Bug report](#) discussing loss of precision when parsing ISO8601 timestamps