datetimerange Documentation

Release 2.1.0

Author

Feb 19, 2023

TABLE OF CONTENTS

CHAPTER

ONE

DATETIMERANGE

1.1 Summary

DateTimeRange is a Python library to handle a time range. e.g. check whether a time is within the time range, get the intersection of time ranges, truncate a time range, iterate through a time range, and so forth.

CHAPTER

TWO

INSTALLATION

2.1 Installation: pip

```
pip install DateTimeRange
```

2.2 Installation: conda

```
conda install -c conda-forge datetimerange
```

CHAPTER

THREE

DEPENDENCIES

- Python 3.7+
- Python package dependencies (automatically installed)

CHAPTER

FOUR

EXAMPLES

`datetime.datetime` instance can be used as an argument value as well as time-string in the following examples.

Note: Use not the DST (Daylight Saving Time) offset, but the standard time offset when you use datetime string as an argument. *DateTimeRange* class automatically calculate daylight saving time. Some examples are below

```
>>>from datetimerange import DateTimeRange
>>>time_range = DateTimeRange("2015-03-08T00:00:00-0400", "2015-03-08T12:00:00-
↪0400")
>>>time_range.timedelta
datetime.timedelta(0, 39600)  # 11 hours
```

```
>>>from datetimerange import DateTimeRange
>>>time_range = DateTimeRange("2015-11-01T00:00:00-0400", "2015-11-01T12:00:00-
↪0400")
>>>time_range.timedelta
datetime.timedelta(0, 46800)  # 13 hours
```

4.1 Create a DateTimeRange instance from start and end datetime

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
str(time_range)
```

Output

```
'2015-03-22T10:00:00+0900 - 2015-03-22T10:10:00+0900'
```

4.2 Compare time ranges

Sample Code

```
from datetimerange import DateTimeRange
lhs = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-22T10:10:00+0900")
rhs = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-22T10:10:00+0900")
print("lhs == rhs: ", lhs == rhs)
print("lhs != rhs: ", lhs != rhs)
```

Output

```
lhs == rhs:  True
lhs != rhs:  False
```

4.3 Move the time range

Sample Code

```
import datetime
from datetimerange import DateTimeRange
value = DateTimeRange("2015-03-22T10:10:00+0900", "2015-03-22T10:20:00+0900
↪")
print(value + datetime.timedelta(seconds=10 * 60))
print(value - datetime.timedelta(seconds=10 * 60))
```

Output

```
2015-03-22T10:20:00+0900 - 2015-03-22T10:30:00+0900
2015-03-22T10:00:00+0900 - 2015-03-22T10:10:00+0900
```

4.4 Change string conversion format

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
time_range.start_time_format = "%Y/%m/%d"
time_range.end_time_format = "%Y/%m/%dT%H:%M:%S%z"
time_range
```

Output

```
2015/03/22 - 2015/03/22T10:10:00+0900
```

4.5 Add elapsed time when conversion to string

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
time_range.is_output_elapse = True
time_range
```

Output

```
2015-03-22T10:00:00+0900 - 2015-03-22T10:10:00+0900 (0:10:00)
```

4.6 Change separator of the converted string

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
time_range.separator = " to "
time_range
```

Output

```
2015-03-22T10:00:00+0900 to 2015-03-22T10:10:00+0900
```

4.7 Get start time as datetime.datetime

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
time_range.start_datetime
```

Output

```
datetime.datetime(2015, 3, 22, 10, 0, tzinfo=tzoffset(None, 32400))
```

4.8 Get start time as string (formatted with start_time_format)

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
print(time_range.get_start_time_str())
time_range.start_time_format = "%Y/%m/%d %H:%M:%S"
print(time_range.get_start_time_str())
```

Output

```
2015-03-22T10:00:00+0900
2015/03/22 10:00:00
```

4.9 Get end time as datetime.datetime

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
time_range.end_datetime
```

Output

```
datetime.datetime(2015, 3, 22, 10, 10, tzinfo=tzoffset(None, 32400))
```

4.10 Get end time as string (formatted with end_time_format)

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
print(time_range.get_end_time_str())
time_range.end_time_format = "%Y/%m/%d %H:%M:%S"
print(time_range.get_end_time_str())
```

Output

```
2015-03-22T10:10:00+0900
2015/03/22 10:10:00
```

4.11 Get datetime.timedelta (from start_datetime to the end_datetime)

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
time_range.timedelta
```

Output

```
datetime.timedelta(0, 600)
```

4.12 Get timedelta as seconds (from start_datetime to the end_datetime)

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
time_range.get_timedelta_second()
```

Output

```
600.0
```

4.13 Get an iterator

Sample Code 1

```
import datetime
from datetimerange import DateTimeRange

time_range = DateTimeRange("2015-01-01T00:00:00+0900", "2015-01-
↪04T00:00:00+0900")
for value in time_range.range(datetime.timedelta(days=1)):
    print(value)
```

Output 1

```
2015-01-01 00:00:00+09:00
2015-01-02 00:00:00+09:00
2015-01-03 00:00:00+09:00
2015-01-04 00:00:00+09:00
```

Sample Code 2

```
from datetimerange import DateTimeRange
from dateutil.relativedelta import relativedelta

time_range = DateTimeRange("2015-01-01T00:00:00+0900", "2016-01-
↪01T00:00:00+0900")
for value in time_range.range(relativedelta(months=+4)):
    print(value)
```

Output 2

```
2015-01-01 00:00:00+09:00
2015-05-01 00:00:00+09:00
2015-09-01 00:00:00+09:00
2016-01-01 00:00:00+09:00
```

4.14 Set start time

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange()
print(time_range)
time_range.set_start_datetime("2015-03-22T10:00:00+0900")
print(time_range)
```

Output

```
NaT - NaT
2015-03-22T10:00:00+0900 - NaT
```

4.15 Set end time

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange()
print(time_range)
time_range.set_end_datetime("2015-03-22T10:10:00+0900")
print(time_range)
```

Output

```
NaT - NaT
NaT - 2015-03-22T10:10:00+0900
```

4.16 Set time range (set both start and end time)

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange()
print(time_range)
time_range.set_time_range("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
print(time_range)
```

Output

```
NaT - NaT
2015-03-22T10:00:00+0900 - 2015-03-22T10:10:00+0900
```

4.17 Test whether the time range is set

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange()
print(time_range.is_set())
time_range.set_time_range("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
print(time_range.is_set())
```

Output

```
False
True
```

4.18 Validate time inversion

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange("2015-03-22T10:10:00+0900", "2015-03-
↪22T10:00:00+0900")
try:
    time_range.validate_time_inversion()
except ValueError:
    print("time inversion")
```

Output

```
time inversion
```

4.19 Test whether the time range is valid

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange()
print(time_range.is_valid_timerange())
time_range.set_time_range("2015-03-22T10:20:00+0900", "2015-03-
↪22T10:10:00+0900")
print(time_range.is_valid_timerange())
time_range.set_time_range("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
print(time_range.is_valid_timerange())
```

Output

```
False
False
True
```

4.20 Test whether a value within the time range

Sample Code

```
from datetimerange import DateTimeRange

time_range = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
print("2015-03-22T10:05:00+0900" in time_range)
print("2015-03-22T10:15:00+0900" in time_range)

time_range_smaller = DateTimeRange("2015-03-22T10:03:00+0900", "2015-03-
↪22T10:07:00+0900")
print(time_range_smaller in time_range)
```

Output

```
True
False
True
```

4.21 Test whether a value intersects the time range

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
x = DateTimeRange("2015-03-22T10:05:00+0900", "2015-03-22T10:15:00+0900")
time_range.is_intersection(x)
```

Output

```
True
```

4.22 Make an intersected time range

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
x = DateTimeRange("2015-03-22T10:05:00+0900", "2015-03-22T10:15:00+0900")
time_range.intersection(x)
```

Output

```
2015-03-22T10:05:00+0900 - 2015-03-22T10:10:00+0900
```

4.23 Make an subtracted time range

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange("2015-01-22T09:50:00+0900", "2015-01-
↪22T10:00:00+0900")
x = DateTimeRange("2015-01-22T09:55:00+0900", "2015-01-22T09:56:00+0900")
time_range.subtract(x)
```

Output

```
[2015-01-22T09:50:00+0900 - 2015-01-22T09:55:00+0900,
 2015-01-22T09:56:00+0900 - 2015-01-22T10:00:00+0900]
```

4.24 Make an encompassed time range

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
x = DateTimeRange("2015-03-22T10:05:00+0900", "2015-03-22T10:15:00+0900")
time_range.encompass(x)
```

Output

```
2015-03-22T10:00:00+0900 - 2015-03-22T10:15:00+0900
```

4.25 Truncate time range

Sample Code

```
from datetimerange import DateTimeRange

time_range = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
time_range.is_output_elapse = True
print("before truncate: ", time_range)

time_range.truncate(10)
print("after truncate:  ", time_range)
```

Output

```
before truncate:  2015-03-22T10:00:00+0900 - 2015-03-22T10:10:00+0900␣
↪(0:10:00)
after truncate:   2015-03-22T10:00:30+0900 - 2015-03-22T10:09:30+0900␣
↪(0:09:00)
```

CHAPTER

FIVE

REFERENCE

5.1 DateTimeRange class

class datetimerange.**DateTimeRange**(*start_datetime: Optional[Union[datetime, str]] = None*, *end_datetime: Optional[Union[datetime, str]] = None*, *start_time_format: str = '%Y-%m-%dT%H:%M:%S%z'*, *end_time_format: str = '%Y-%m-%dT%H:%M:%S%z'*)

Bases: `object`

A class that represents a range of datetime.

Parameters

- **start_datetime** (*datetime.datetime/str*) – Value to set to the **start** time of the time range. There are three types that acceptable as an input value: **(1)** `datetime.datetime` object. **(2)** datetime string: e.g. `"2017-01-22T04:56:00+0900"`. **(3)** timestamp (`str`/`int`): e.g. `1485685623`.
- **end_datetime** (*datetime.datetime/str*) – Value to set to the **end** time of the time range. There are three types that acceptable as an input value: **(1)** `datetime.datetime` object. **(2)** datetime string: e.g. `"2017-01-22T04:56:00+0900"`. **(3)** timestamp (`str`/`int`): e.g. `1485685623`.

Examples

Sample Code

```
from datetimerange import DateTimeRange
DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
```

Output

```
2015-03-22T10:00:00+0900 - 2015-03-22T10:10:00+0900
```

start_time_format: `str` = `"%Y-%m-%dT%H:%M:%S%z"`

Conversion format string for *start_datetime*.

See also:

get_start_time_str()

end_time_format: `str` = `"%Y-%m-%dT%H:%M:%S%z"`

Conversion format string for *end_datetime*.

See also:

get_end_time_str()

__contains__(*x: Union[datetime, DateTimeRange, str]*) → bool

Parameters
x (datetime.datetime/DateTimeRange/str) – datetime.datetime/DateTimeRange instance to compare. Parse and convert to datetime.datetime if the value type is str.

Returns
True if the **x** is within the time range

Return type
bool

Sample Code

```
from datetimerange import DateTimeRange

time_range = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
print("2015-03-22T10:05:00+0900" in time_range)
print("2015-03-22T10:15:00+0900" in time_range)

time_range_smaller = DateTimeRange("2015-03-22T10:03:00+0900", "2015-
↪03-22T10:07:00+0900")
print(time_range_smaller in time_range)
```

Output

```
True
False
True
```

See also:

validate_time_inversion()

__eq__(*other: object*) → bool

Return self==value.

__hash__ = None

__init__(*start_datetime: Optional[Union[datetime, str]] = None*, *end_datetime: Optional[Union[datetime, str]] = None*, *start_time_format: str = '%Y-%m-%dT%H:%M:%S%z'*, *end_time_format: str = '%Y-%m-%dT%H:%M:%S%z'*) → None

__ne__(*other: object*) → bool

Return self!=value.

__repr__() → str

Return repr(self).

__weakref__

list of weak references to the object (if defined)

encompass(*x:* DateTimeRange) → *DateTimeRange*

Create a new time range that encompasses the input and the current time range.

Parameters
x (DateTimeRange) – Value to compute encompass with the current time range.

Sample Code

```
from datetimerange import DateTimeRange
dtr0 = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
dtr1 = DateTimeRange("2015-03-22T10:05:00+0900", "2015-03-
↪22T10:15:00+0900")
dtr0.encompass(dtr1)
```

Output

```
2015-03-22T10:00:00+0900 - 2015-03-22T10:15:00+0900
```

property end_datetime: Optional[datetime]

End time of the time range. :rtype: Optional[datetime.datetime]

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
time_range.end_datetime
```

Output

```
datetime.datetime(2015, 3, 22, 10, 10, tzinfo=tzoffset(None, 32400))
```

Type
return

classmethod from_range_text(*range_text: str*, *separator: str = '\\s+\\-\\s+'*, *start_time_format: Optional[str] = None*, *end_time_format: Optional[str] = None*) → *DateTimeRange*

Create a DateTimeRange instance from a datetime range text.

Parameters

- **range_text** (*str*) – Input text that includes datetime range. e.g. 2021-01-23T10:00:00+0400 - 2021-01-232T10:10:00+0400
- **separator** (*str*) – Regular expression that separating the range_text to start and end time.

Returns
DateTimeRange Created instance.

get_end_time_str() → str

Returns
end_datetime as a str formatted with *end_time_format*. Return NOT_A_TIME_STR if invalid datetime or format.

Return type
str

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
print(time_range.get_end_time_str())
time_range.end_time_format = "%Y/%m/%d %H:%M:%S"
print(time_range.get_end_time_str())
```

Output

```
2015-03-22T10:10:00+0900
2015/03/22 10:10:00
```

get_start_time_str() → str

Returns

start_datetime as `str` formatted with *start_time_format*. Return `NOT_A_TIME_STR` if the invalid value or the invalid format.

Return type

str

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
print(time_range.get_start_time_str())
time_range.start_time_format = "%Y/%m/%d %H:%M:%S"
print(time_range.get_start_time_str())
```

Output

```
2015-03-22T10:00:00+0900
2015/03/22 10:00:00
```

get_timedelta_second() → float

Returns

(*end_datetime* - *start_datetime*) as seconds

Return type

float

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
time_range.get_timedelta_second()
```

Output

```
600.0
```

intersection(*x:* DateTimeRange, *intersection_threshold: Optional[Union[timedelta, relativedelta]] = None*) → *DateTimeRange*

Create a new time range that overlaps the input and the current time range. If no overlaps found, return a time range that set `None` for both start and end time.

Parameters

- **x** (DateTimeRange) – Value to compute intersection with the current time range.
- **intersection_threshold** (*Union[datetime.timedelta, dateutil.relativedelta.relativedelta, None]*) – Minimum time constraint that an intersection must have. Defaults to `None` (no constraint).

Sample Code

```
from datetimerange import DateTimeRange
dtr0 = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
dtr1 = DateTimeRange("2015-03-22T10:05:00+0900", "2015-03-
↪22T10:15:00+0900")
dtr0.intersection(dtr1)
```

Output

```
2015-03-22T10:05:00+0900 - 2015-03-22T10:10:00+0900
```

is_intersection(*x:* DateTimeRange, *intersection_threshold: Optional[Union[timedelta, relativedelta]] = None*) → bool

Parameters

- **x** (DateTimeRange) – Value to compare
- **intersection_threshold** (*Union[datetime.timedelta, dateutil.relativedelta.relativedelta, None]*) – Minimum time constraint that an intersection must have. Defaults to `None` (no constraint).

Returns

True if intersect with `x`

Return type

bool

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
x = DateTimeRange("2015-03-22T10:05:00+0900", "2015-03-
↪22T10:15:00+0900")
time_range.is_intersection(x)
```

Output

```
True
```

is_set() → bool

Returns

True if both *start_datetime* and *end_datetime* were not `None`.

Return type
bool

Sample Code

```
from datetimerange import DateTimeRange

time_range = DateTimeRange()
print(time_range.is_set())

time_range.set_time_range("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
print(time_range.is_set())
```

Output

```
False
True
```

`is_valid_timerange`() → bool

Returns
`True` if the time range is not null and not time inversion.

Return type
bool

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange()
print(time_range.is_valid_timerange())
time_range.set_time_range("2015-03-22T10:20:00+0900", "2015-03-
↪22T10:10:00+0900")
print(time_range.is_valid_timerange())
time_range.set_time_range("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
print(time_range.is_valid_timerange())
```

Output

```
False
False
True
```

See also:

is_set() validate_time_inversion()

`range`(*step: Union[timedelta, relativedelta]*) → Iterator[datetime]

Return an iterator object.

Parameters
step (datetime.timedelta/dateutil.relativedelta.relativedelta) – Step of iteration.

Returns
iterator

Return type
iterator

Sample Code

```
import datetime
from datetimerange import DateTimeRange

time_range = DateTimeRange("2015-01-01T00:00:00+0900", "2015-01-
↪04T00:00:00+0900")
for value in time_range.range(datetime.timedelta(days=1)):
    print(value)
```

Output

```
2015-01-01 00:00:00+09:00
2015-01-02 00:00:00+09:00
2015-01-03 00:00:00+09:00
2015-01-04 00:00:00+09:00
```

set_end_datetime(*value: Optional[Union[datetime, str]]*, *timezone: Optional[str] = None*) → None

Set the end time of the time range.

Parameters
value (*datetime.datetime/str*) – Value to set to the **end** time of the time range. There are three types that acceptable as an input value: **(1)** `datetime.datetime` object. **(2)** datetime string: e.g. `"2017-01-22T04:56:00+0900"`. **(3)** timestamp (`str`/`int`): e.g. `1485685623`.

Raises
`ValueError` – If the value is invalid as a `datetime.datetime` value.

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange()
print(time_range)
time_range.set_end_datetime("2015-03-22T10:10:00+0900")
print(time_range)
```

Output

```
NaT - NaT
NaT - 2015-03-22T10:10:00+0900
```

set_start_datetime(*value: Optional[Union[datetime, str]]*, *timezone: Optional[str] = None*) → None

Set the start time of the time range.

Parameters
value (`datetime.datetime`/`str`) – Value to set to the **start** time of the time range. There are three types that acceptable as an input value: **(1)** `datetime.datetime` object. **(2)** datetime string: e.g. `"2017-01-22T04:56:00+0900"`. **(3)** timestamp (`str`/`int`): e.g. `1485685623`.

Raises
`ValueError` – If the value is invalid as a `datetime.datetime` value.

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange()
print(time_range)
time_range.set_start_datetime("2015-03-22T10:00:00+0900")
print(time_range)
```

Output

```
NaT - NaT
2015-03-22T10:00:00+0900 - NaT
```

set_time_range(*start: Optional[Union[datetime, str]]*, *end: Optional[Union[datetime, str]]*) → None

Parameters

- **start** (*datetime.datetime/str*) – Value to set to the **start** time of the time range. There are three types that acceptable as an input value: **(1)** `datetime.datetime` object. **(2)** datetime string: e.g. `"2017-01-22T04:56:00+0900"`. **(3)** timestamp (`str`/`int`): e.g. `1485685623`.
- **end** (*datetime.datetime/str*) – Value to set to the **end** time of the time range. There are three types that acceptable as an input value: **(1)** `datetime.datetime` object. **(2)** datetime string: e.g. `"2017-01-22T04:56:00+0900"`. **(3)** timestamp (`str`/`int`): e.g. `1485685623`.

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange()
print(time_range)
time_range.set_time_range("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
print(time_range)
```

Output

```
NaT - NaT
2015-03-22T10:00:00+0900 - 2015-03-22T10:10:00+0900
```

split(*separator: Union[str, datetime]*) → List[*DateTimeRange*]

Split the DateTimerange in two DateTimerange at a specific datetime.

Parameters

separator (*Union[str, datetime.datetime]*) – Date and time to split the DateTimeRange. This value will be included for both of the ranges after split.

Sample Code

```
from datetimerange import DateTimeRange
dtr = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
dtr.split("2015-03-22T10:05:00+0900")
```

Output

```
[2015-03-22T10:00:00+0900 - 2015-03-22T10:05:00+0900,
2015-03-22T10:05:00+0900 - 2015-03-22T10:10:00+0900]
```

property start_datetime: Optional[datetime]

Start time of the time range. :rtype: Optional[datetime.datetime]

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
time_range.start_datetime
```

Output

```
datetime.datetime(2015, 3, 22, 10, 0, tzinfo=tzoffset(None, 32400))
```

Type
return

subtract(*x:* DateTimeRange) → List[*DateTimeRange*]

Remove a time range from this one and return the result.

- The result will be `[self.copy()]` if the second range does not overlap the first
- The result will be `[]` if the second range wholly encompasses the first range
- The result will be `[new_range]` if the second range overlaps one end of the range
- The result will be `[new_range1, new_range2]` if the second range is an internal sub range of the first

Parameters
x (DateTimeRange) – Range to remove from this one.

Returns
List(DateTimeRange) List of new ranges when the second range is removed from this one

Sample Code

```
from datetimerange import DateTimeRange
dtr0 = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
dtr1 = DateTimeRange("2015-03-22T10:05:00+0900", "2015-03-
↪22T10:15:00+0900")
dtr0.subtract(dtr1)
```

Output

```
[2015-03-22T10:00:00+0900 - 2015-03-22T10:05:00+0900]
```

property timedelta: timedelta

Returns
(*end_datetime* - *start_datetime*) as datetime.timedelta

Return type
datetime.timedelta

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange("2015-03-22T10:00:00+0900", "2015-03-
↪22T10:10:00+0900")
time_range.timedelta
```

Output

```
datetime.timedelta(0, 600)
```

truncate(*percentage: float*) → None

Truncate `percentage` / 2 [%] of whole time from first and last time.

Parameters

percentage (*float*) – Percentage of truncate.

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange(
    "2015-03-22T10:00:00+0900", "2015-03-22T10:10:00+0900")
time_range.is_output_elapse = True
print(time_range)
time_range.truncate(10)
print(time_range)
```

Output

```
2015-03-22T10:00:00+0900 - 2015-03-22T10:10:00+0900 (0:10:00)
2015-03-22T10:00:30+0900 - 2015-03-22T10:09:30+0900 (0:09:00)
```

validate_time_inversion() → None

Check time inversion of the time range.

Raises

- **ValueError** – If *start_datetime* is bigger than *end_datetime*.
- **TypeError** – Any one of *start_datetime* and *end_datetime*, or both is inappropriate datetime value.

Sample Code

```
from datetimerange import DateTimeRange
time_range = DateTimeRange("2015-03-22T10:10:00+0900", "2015-03-
↪22T10:00:00+0900")
try:
    time_range.validate_time_inversion()
except ValueError:
    print("time inversion")
```

Output

```
time inversion
```

CHAPTER

SIX

CHANGELOG

https://github.com/thombashi/DateTimeRange/releases

CHAPTER

SEVEN

SPONSORS

Become a sponsor

CHAPTER

EIGHT

INDICES AND TABLES

- genindex

CHAPTER

NINE

LINKS

- GitHub repository
- Issue tracker
- pip: A tool for installing python packages

CHAPTER

TEN

INDICES AND TABLES

- genindex

INDEX

Symbols

D

E

F

G

I

R

S

T

V