python-mobilitydb Documentation

Release 0.01

Esteban Zimányi

Jul 24, 2020

CONTENTS:

`python-mobilitydb` is a database adapter to access MobilityDB from Python. It supports both the psycopg2 and the asyncpg adapters for PostgreSQL and uses the postgis adapter for PostGIS.

CHAPTER ONE

INSTALLATION

1.1 Requirements

`python-mobilitydb` has several dependencies beyond an installation of Python 3.x:

- psycopg2 or asyncpg to connect to PostgreSQL,
- postgis to connect to PostGIS,
- Spans for an implementation of PostgreSQL's range types,
- python-dateutil for extensions to the standard datetime module,
- parsec for parsing.

1.2 Python Package Index

`python-mobilitydb` may be installed from PyPI.

```
$ pip install python-mobilitydb
```

1.3 Source

The package sources are available at https://github.com/ULB-CoDE-WIT/python-mobilitydb. Building and installing `python-mobilitydb` from source can be done with setuptools:

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

1.3.1 Tests

Tests require pytest and pytest-asyncio.

```
$ pytest
```

The PostgreSQL database server must be started before launching the tests.

1.3.2 Documentation

Building the documentation from source requires Sphinx. By default, the documentation will be rendered in HTML:

```
$ python setup.py build_sphinx
```

For other documentation output formats, see the options in the `docs` subdirectory:

```
$ cd docs
$ make
```

CHAPTER

TWO

BASIC USAGE

`python-mobilitydb` is a Python converter to and from the temporal types provided by MobilityDB, that is `tbool`, `tint`, `tfloat`, `ttext`, `tgeompoint`, and `tgeogpoint`.

2.1 `TBool`, `TInt`, and `TText`

Classes *`TBool`*, *`TInt`*, and *`TText`* represent, respectively, temporal Booleans, temporal integers, and temporal strings. These classes have in common that their base type is discrete. As a consequence of this, the interpolation for the instances of sequence or sequence set duration is stepwise. We illustrate next how to create new instances of the `TInt` class, the creation of instances of the `TBool` and `TText` classes is similar.

New *`TInt`* instances can be created by using one of its subclasses *`TIntInst`*, *`TIntI`*, *`TIntSeq`*, or *`TIntS`*.

New *`TIntInst`* instances can be created either with a single string argument as in MobilityDB or with two arguments: the value and the timestamp.

```
>>> from dateutil.parser import parse
>>> from mobilitydb import TIntInst
>>> TIntInst("1@2020-01-01 00:00:00+01")
>>> TIntInst("1", "2020-01-01 00:00:00+01")
>>> TIntInst(1, parse("2020-01-01 00:00:00+01"))
```

New *`TIntI`* instances can be created either with a single string argument as in MobilityDB or with a tuple or list of the composing instants.

```
>>> from mobilitydb import TIntI
>>> TIntI("{1@2020-01-01, 2@2020-01-02}")
>>> TIntI(["1@2020-01-01", "2@2020-01-02"])
>>> TIntI("1@2020-01-01", "2@2020-01-02")
>>> TIntI(TIntInst(1, "2020-01-01"), TIntInst(2, "2020-01-02"))
```

New *`TIntSeq`* instances can be created either with a single string argument as in MobilityDB or with several arguments: the list of composing instants, the left inclusion flag, and the right inclusion flag, where only the first argument is mandatory.

```
>>> from mobilitydb import TIntSeq
>>> TIntSeq("[1@2020-01-01, 2@2020-01-02]")
>>> TIntSeq(["1@2020-01-01", "2@2020-01-02"], lower_inc= True, upper_inc=True)
>>> TIntSeq([TIntInst(1, "2020-01-01"), TIntInst(2, "2020-01-02")], lower_inc= True,␣
↪upper_inc=True)
```

Finally, new *`TIntS`* instances can be created either with a single string argument as in MobilityDB or with a single argument: the list of composing sequences.

```
>>> from mobilitydb import TIntS
>>> TIntS("{[1@2020-01-01, 2@2020-01-02], [2@2020-01-03, 1@2020-01-04]}")
>>> TIntS(["[1@2020-01-01, 2@2020-01-02]", "[2@2020-01-03, 1@2020-01-04]"])
>>> TIntS([TIntSeq("[1@2020-01-01, 2@2020-01-02]"), TIntSeq("[2@2020-01-03, 1@2020-01-
↪04]")])
```

2.2 TFloat

Class *TFloat* represents temporal floats. Since the base type of `TFloat` is continuous, the interpolation for instances of the sequence or sequence set duration may be either linear or stepwise, the former being the default.

New *TFloat* instances can be created by using one of its subclasses *TFloatInst*, *TFloatI*, *TFloatSeq*, or *TFloatS*.

New *TFloatInst* instances can be created either with a single string argument as in MobilityDB or with two arguments: the value and the timestamp.

```
>>> from dateutil.parser import parse
>>> from mobilitydb import TFloatInst
>>> TFloatInst("1.0@2020-01-01 00:00:00+01")
>>> TFloatInst("1.0", "2020-01-01 00:00:00+01")
>>> TFloatInst(1.0, parse("2020-01-01 00:00:00+01"))
```

New *TFloatI* instances can be created either with a single string argument as in MobilityDB or with a tuple or list of the composing instants.

```
>>> from mobilitydb import TFloatI
>>> TFloatI("{1.0@2020-01-01, 2.0@2020-01-02}")
>>> TFloatI(["1.0@2020-01-01", "2.0@2020-01-02"])
>>> TFloatI("1.0@2020-01-01", "2.0@2020-01-02")
>>> TFloatI(TFloatInst("1.0@2020-01-01"), TFloatInst("2.0@2020-01-02"))
```

New *TFloatSeq* instances can be created either with a single string argument as in MobilityDB or with several arguments: the list of composing instants, the left inclusion flag, the right inclusion flag, and the interpolation, where only the first argument is mandatory.

```
>>> from mobilitydb import TFloatSeq
>>> TFloatSeq("[1.0@2020-01-01, 2.0@2020-01-02]")
>>> TFloatSeq("Interp=Stepwise;[1.0@2020-01-01, 2.0@2020-01-02]")
>>> TFloatSeq(["1.0@2020-01-01", "2.0@2020-01-02"], lower_inc= True, upper_inc=True,␣
↪interp='Stepwise')
```

Finally, new *TFloatS* instances can be created either with a single string argument as in MobilityDB or with two arguments: the list of composing sequences and the interpolation, where only the first argument is mandatory.

```
>>> from mobilitydb import TFloatS
>>> TFloatS("{[1.0@2020-01-01, 2.0@2020-01-02], [2.0@2020-01-03, 1.0@2020-01-04]}")
>>> TFloatS(["[1.0@2020-01-01, 2.0@2020-01-02]", "[2.0@2020-01-03, 1.0@2020-01-04]"],␣
↪interp='Stepwise')
```

2.3 TGeomPoint and TGeogPoint

Class *TGeomPoint* represents temporal geometric points with Cartesian (planar) coordinates while *TGeogPoint* represents geographic points with geodetic (spherical) coordinates. Since the base type of these classes is continuous, the interpolation for the instances of sequence or sequence set duration may be either linear or stepwise, the former being the default. We illustrate next how to create instances of the TGeomPoint class, the creation of instances of the TGeogPoint class is similar.

New *TGeomPoint* instances can be created by using one of its subclasses *TGeomPointInst*, *TGeomPointI*, *TGeomPointSeq*, or *TGeomPointS*.

New *TGeomPointInst* instances can be created either with a single string argument as in MobilityDB or with several arguments: the value, the timestamp, and the SRID, the latter being optional. In both cases, the value of the point can be specified using a Well-Known Text (WKT) or Well-Known Binary (WKB) representation as well as its format variations Extended Well-Known Text (EWKT) and Extended Well-Known Binary (EWKB).

```
>>> from dateutil.parser import parse
>>> from postgis import Point
>>> from mobilitydb import TGeomPointInst
>>> TGeomPointInst("POINT(1 1)@2020-01-01 00:00:00+01")
>>> TGeomPointInst("SRID=4326;POINT(1 1)@2020-01-01 00:00:00+01")
>>> TGeomPointInst("0101000000000000000000004AC000000000000000@2020-01-01")
>>> TGeomPointInst("POINT(1 1)", "2020-01-01 00:00:00+01", srid=4326)
>>> TGeomPointInst(Point(1, 1), parse("2020-01-01 00:00:00+01"), srid=4326)
```

New *TGeomPointI* instances can be created either with a single string argument as in MobilityDB or with two arguments: the list of composing instants and the SRID, the latter being optional.

```
>>> from mobilitydb import TGeomPointI
>>> TGeomPointI("{POINT(1 1)@2020-01-01, POINT(2 2)@2020-01-02}")
>>> TGeomPointI(["POINT(1 1)@2020-01-01", "POINT(2 2)@2020-01-02"], srid=4326)
>>> TGeomPointI([TGeomPointInst("POINT(1 1)@2020-01-01"), TGeomPointInst("POINT(2␣
↪2)@2020-01-02")], srid=4326)
```

New *TGeomPointSeq* instances can be created either with a single string argument as in MobilityDB or with several arguments: the list of composing instants, the left inclusion flag, the right inclusion flag, the interpolation, and the SRID, where only the first argument is mandatory.

```
>>> from mobilitydb import TGeomPointSeq
>>> TGeomPointSeq("[POINT(1 1)@2020-01-01, POINT(2 2)@2020-01-02]")
>>> TGeomPointSeq("SRID=4326;[POINT(1 1)@2020-01-01, POINT(2 2)@2020-01-02]")
>>> TGeomPointSeq("SRID=4326,Interp=Stepwise;[POINT(1 1)@2020-01-01, POINT(2 2)@2020-
↪01-02]")
>>> TGeomPointSeq(["POINT(1 1)@2020-01-01", "POINT(2 2)@2020-01-02"], lower_inc= True,
↪ upper_inc=True, interp='Stepwise', srid=4326)
>>> TGeomPointSeq([TGeomPointInst("POINT(1 1)@2020-01-01"), TGeomPointInst("POINT(2␣
↪2)@2020-01-02")], lower_inc= True, upper_inc=True, interp='Stepwise', srid=4326)
```

Finally, new *TGeomPointS* instances can be created either with a single string argument as in MobilityDB or with several arguments: the list of composing sequences, the interpolation, and the SRID, where only the first argument is mandatory.

```
>>> from mobilitydb import TGeomPointS
>>> TGeomPointS("{[POINT(1 1)@2020-01-01, POINT(2 2)@2020-01-02], [POINT(2 2)@2020-01-
↪03, POINT(1 1)@2020-01-04]}")
>>> TGeomPointS("SRID=4326;{[POINT(1 1)@2020-01-01, POINT(2 2)@2020-01-02], [POINT(2␣
↪2)@2020-01-03, POINT(1 1)@2020-01-04]}")
```

(continues on next page)

(continued from previous page)

```
>>> TGeomPointS(["[POINT(1 1)@2020-01-01, POINT(2 2)@2020-01-02]", "[POINT(2 2)@2020-
↪01-03, POINT(1 1)@2020-01-04]"], interp='Stepwise', srid=4326)
>>> TGeomPointS([TGeomPointSeq("[POINT(1 1)@2020-01-01, POINT(2 2)@2020-01-02]"),␣
↪TGeomPointSeq("[POINT(2 2)@2020-01-03, POINT(1 1)@2020-01-04]")], interp='Stepwise',
↪ srid=4326)
```

CHAPTER

THREE

API REFERENCE

3.1 Time Types

class mobilitydb.time.**Period**(*lower*, *upper=None*, *lower_inc=None*, *upper_inc=None*)

Class for representing sets of contiguous timestamps between a lower and an upper bound. The bounds may be inclusive or not.

`Period` objects can be created with a single argument of type string as in MobilityDB.

```
>>> Period('(2019-09-08 00:00:00+01, 2019-09-10 00:00:00+01)')
```

Another possibility is to give a tuple of arguments as follows:

- `lower` and `upper` are instances of `str` or `datetime` specifying the bounds,
- `lower_inc` and `upper_inc` are instances of `bool` specifying whether the bounds are inclusive or not. By default, `lower_inc` is `True` and `upper_inc` is `False`.

Some examples are given next.

```
>>> Period('2019-09-08 00:00:00+01', '2019-09-10 00:00:00+01')
>>> Period('2019-09-08 00:00:00+01', '2019-09-10 00:00:00+01', False, True)
>>> Period(parse('2019-09-08 00:00:00+01'), parse('2019-09-10 00:00:00+01'))
>>> Period(parse('2019-09-08 00:00:00+01'), parse('2019-09-10 00:00:00+01'),␣
↪False, True)
```

lower
Lower bound

upper
Upper bound

lower_inc
Is the lower bound inclusive?

upper_inc
Is the upper bound inclusive?

timespan
Time interval on which the period is defined

shift(*timedelta*)
Shift the period by a time interval

overlap(*other*)
Do the periods share a timestamp?

contains_timestamp(*datetime*)
Does the period contain the timestamp?

class mobilitydb.time.**TimestampSet**(**argv*)
Class for representing lists of distinct timestamp values.

`TimestampSet` objects can be created with a single argument of type string as in MobilityDB.

```
>>> TimestampSet('{2019-09-08 00:00:00+01, 2019-09-10 00:00:00+01, 2019-09-11␣
↪00:00:00+01}')
```

Another possibility is to give a tuple or list of composing timestamps, which can be instances of `str` or `datetime`. The composing timestamps must be given in increasing order.

```
>>> TimestampSet(['2019-09-08 00:00:00+01', '2019-09-10 00:00:00+01', '2019-09-11␣
↪00:00:00+01'])
>>> TimestampSet([parse('2019-09-08 00:00:00+01'), parse('2019-09-10 00:00:00+01
↪'), parse('2019-09-11 00:00:00+01')])
>>> TimestampSet('2019-09-08 00:00:00+01', '2019-09-10 00:00:00+01', '2019-09-11␣
↪00:00:00+01')
>>> TimestampSet(parse('2019-09-08 00:00:00+01'), parse('2019-09-10 00:00:00+01'),
↪ parse('2019-09-11 00:00:00+01'))
```

period
Period on which the timestamp set is defined ignoring the potential time gaps

numTimestamps
Number of timestamps

startTimestamp
Start timestamp

endTimestamp
End timestamp

timestampN(*n*)
N-th timestamp

timestamps
Distinct timestamps

shift(*timedelta*)
Shift the timestamp set by a time interval

class mobilitydb.time.**PeriodSet**(**argv*)
Class for representing lists of disjoint periods.

`PeriodSet` objects can be created with a single argument of type string as in MobilityDB.

```
>>> PeriodSet('{[2019-09-08 00:00:00+01, 2019-09-10 00:00:00+01], [2019-09-11␣
↪00:00:00+01, 2019-09-12 00:00:00+01]}')
```

Another possibility is to give a list or tuple specifying the composing periods, which can be instances of `str` or `Period`. The composing periods must be given in increasing order.

```
>>> PeriodSet(['[2019-09-08 00:00:00+01, 2019-09-10 00:00:00+01]', '[2019-09-11␣
↪00:00:00+01, 2019-09-12 00:00:00+01]'])
>>> PeriodSet([Period('[2019-09-08 00:00:00+01, 2019-09-10 00:00:00+01]'), Period(
↪'[2019-09-11 00:00:00+01, 2019-09-12 00:00:00+01]')])
```

(continues on next page)

(continued from previous page)

```
>>> PeriodSet('[2019-09-08 00:00:00+01, 2019-09-10 00:00:00+01]', '[2019-09-11␣
↪00:00:00+01, 2019-09-12 00:00:00+01]')
>>> PeriodSet(Period('[2019-09-08 00:00:00+01, 2019-09-10 00:00:00+01]'), Period(
↪'[2019-09-11 00:00:00+01, 2019-09-12 00:00:00+01]'))
```

timespan
: Time interval on which the period set is defined

period
: Period on which the period set is defined ignoring the potential time gaps

numTimestamps
: Number of distinct timestamps

startTimestamp
: Start timestamp

endTimestamp
: End timestamp

timestampN(*n*)
: N-th distinct timestamp

timestamps
: Distinct timestamps

numPeriods
: Number of periods

startPeriod
: Start period

endPeriod
: End period

periodN(*n*)
: N-th period

periods
: Periods

shift(*timedelta*)
: Shift the period set by a time interval

3.2 Temporal Types

class `mobilitydb.temporal.`**Temporal**
: Bases: `object`

Abstract class for representing temporal values of any duration.

BaseClass = None
: Class of the base type, for example, `float` for `TFloat`

BaseClassDiscrete = None
: Boolean value that states whether the base type is discrete or not, for example, `True` for `int` and `False` for `float`

`ComponentClass = None`
Class of the components, for example,

1. `TFloatInst` for both `TFloatI` and `TFloatSeq`
2. `TFloatSeq` for `TFloatS`.

`classmethod duration``()`
Duration of the temporal value, that is, one of `'Instant'`, `'InstantSet'`, `'Sequence'`, or `'SequenceSet'`.

`getValues`
List of distinct values taken by the temporal value.

`startValue`
Start value.

`endValue`
End value.

`minValue`
Minimum value.

`maxValue`
Maximum value.

`getTime`
Period set on which the temporal value is defined.

`timespan`
Interval on which the temporal value is defined.

`period`
Period on which the temporal value is defined ignoring potential time gaps.

`numInstants`
Number of distinct instants.

`startInstant`
Start instant.

`endInstant`
End instant.

`instantN`(*n*)
N-th instant.

`instants`
List of instants.

`numTimestamps`
Number of distinct timestamps.

`startTimestamp`
Start timestamp.

`endTimestamp`
End timestamp.

`timestampN`(*n*)
N-th timestamp.

`timestamps`
List of timestamps.

shift(*timedelta*)
: Shift the temporal value by a time interval

intersectsTimestamp(*datetime*)
: Does the temporal value intersect the timestamp?

intersectsTimestampset(*timestampset*)
: Does the temporal value intersect the timestamp set?

intersectsPeriod(*period*)
: Does the temporal value intersect the period?

intersectsPeriodset(*periodset*)
: Does the temporal value intersect the period set?

class `mobilitydb.temporal.`**TemporalInst**(*value*, *time=None*)

Bases: `mobilitydb.temporal.temporal.Temporal`

Abstract class for representing temporal values of instant duration.

classmethod duration()
: Duration of the temporal value, that is, `'Instant'`.

getValue
: Value component.

getValues
: List of distinct values.

startValue
: Start value.

endValue
: End value.

minValue
: Minimum value.

maxValue
: Maximum value.

getTimestamp
: Timestamp.

getTime
: Period set on which the temporal value is defined.

timespan
: Interval on which the temporal value is defined. It is zero for temporal values of instant duration.

period
: Period on which the temporal value is defined ignoring the potential time gaps.

numInstants
: Number of instants.

startInstant
: Start instant.

endInstant
: End instant.

instantN(*n*)
: N-th instant.

instants
List of instants.

numTimestamps
Number of timestamps.

startTimestamp
Start timestamp.

endTimestamp
End timestamp.

timestampN(*n*)
N-th timestamp

timestamps
List of timestamps.

shift(*timedelta*)
Shift the temporal value by a time interval.

intersectsTimestamp(*timestamp*)
Does the temporal value intersect the timestamp?

intersectsPeriod(*period*)
Does the temporal value intersect the period?

class `mobilitydb.temporal.`**TemporalInstants**
Bases: `mobilitydb.temporal.temporal.Temporal`

Abstract class for representing temporal values of instant set or sequence duration.

getValues
List of distinct values taken by the temporal value.

startValue
Start value.

endValue
End value.

minValue
Minimum value.

maxValue
Maximum value.

numInstants
Number of instants.

startInstant
Start instant.

endInstant
End instant.

instantN(*n*)
N-th instant.

instants
List of instants.

numTimestamps
Number of timestamps.

startTimestamp
Start timestamp.

endTimestamp
End timestamp.

timestampN(*n*)
N-th timestamp.

timestamps
List of timestamps.

shift(*timedelta*)
Shift the temporal value by a time interval.

class `mobilitydb.temporal.`**TemporalI**(**argv*)
Bases: `mobilitydb.temporal.temporalinstants.TemporalInstants`

Abstract class for representing temporal values of instant set duration.

classmethod duration()
Duration of the temporal value, that is, `'InstantSet'`.

getTime
Period set on which the temporal value is defined.

timespan
Interval on which the temporal value is defined. It is zero for temporal values of instant set duration.

period
Period on which the temporal value is defined ignoring the potential time gaps.

intersectsTimestamp(*timestamp*)
Does the temporal value intersect the timestamp?

intersectsPeriod(*period*)
Does the temporal value intersect the period?

class `mobilitydb.temporal.`**TemporalSeq**(*instantList*, *lower_inc=None*, *upper_inc=None*, *interp=None*)
Bases: `mobilitydb.temporal.temporalinstants.TemporalInstants`

Abstract class for representing temporal values of sequence duration.

classmethod duration()
Duration of the temporal value, that is, `'Sequence'`.

lower_inc
Is the lower bound inclusive?

upper_inc
Is the upper bound inclusive?

getTime
Period set on which the temporal value is defined.

timespan
Interval on which the temporal value is defined.

period
Period on which the temporal value is defined.

numSequences
Number of sequences.

startSequence
Start sequence.

endSequence
End sequence.

sequenceN(*n*)
N-th sequence.

sequences
List of sequences.

intersectsTimestamp(*timestamp*)
Does the temporal value intersect the timestamp?

intersectsPeriod(*period*)
Does the temporal value intersect the period?

class `mobilitydb.temporal.`**TemporalS**(*sequenceList*, *interp=None*)
Bases: `mobilitydb.temporal.temporal.Temporal`

Abstract class for representing temporal values of sequence set duration.

classmethod duration()
Duration of the temporal value, that is, `'SequenceSet'`.

getValues
List of distinct values taken by the temporal value.

startValue
Start value.

endValue
End value.

minValue
Minimum value.

maxValue
Maximum value.

getTime
Period set on which the temporal value is defined.

timespan
Interval on which the period set is defined.

period
Period on which the temporal value is defined ignoring the potential time gaps.

numInstants
Number of distinct instants.

startInstant
Start instant.

endInstant
End instant.

instantN(*n*)
N-th distinct instant.

instants
List of instants.

numTimestamps
Number of distinct timestamps.

startTimestamp
Start timestamp.

endTimestamp
End timestamp.

timestampN(*n*)
N-th distinct timestamp.

timestamps
List of timestamps.

numSequences
Number of sequences.

startSequence
Start sequence.

endSequence
End sequence.

sequenceN(*n*)
N-th sequence.

sequences
List of sequences.

shift(*timedelta*)
Shift the temporal value by a time interval.

intersectsTimestamp(*timestamp*)
Does the temporal value intersect the timestamp?

intersectsPeriod(*period*)
Does the temporal value intersect the period?

3.3 Box Types

class `mobilitydb.boxes.`**TBox**(*xmin*, *tmin=None*, *xmax=None*, *tmax=None*)
Bases: `object`

Class for representing bounding boxes with value (`X`) and/or time (`T`) dimensions.

`TBox` objects can be created with a single argument of type string as in MobilityDB.

```
>>> TBox("TBOX((1.0, 2000-01-01), (2.0, 2000-01-02))")
>>> TBox("TBOX((1.0,), (2.0,))")
>>> TBox("TBOX((, 2000-01-01), (, 2000-01-02))")
```

Another possibility is to give the bounds in the following order: `xmin`, `tmin`, `xmax`, `tmax`, where the bounds can be instances of `str`, `float` or `datetime`. All arguments are optional but they must be given in pairs for each dimension and at least one pair must be given.

```
>>> TBox("1.0", "2000-01-01", "2.0", "2000-01-02")
>>> TBox(1.0, 2.0)
>>> TBox(parse("2000-01-01"), parse("2000-01-02"))
```

xmin
Minimum X

tmin
Minimum T

xmax
Maximum X

tmax
Maximum T

class `mobilitydb.boxes.`**STBox**(*bounds*, *dimt=None*, *geodetic=None*, *srid=None*)
Bases: `object`

Class for representing bounding boxes composed of coordinate and/or time dimensions, where the coordinates may be in 2D (`X` and `Y`) or in 3D (`X`, `Y`, and `Z`). For each dimension, minimum and maximum values are stored. The coordinates may be either Cartesian (planar) or geodetic (spherical). Additionally, the SRID of coordinates can be specified.

`STBox` objects can be created with a single argument of type string as in MobilityDB.

```
>>> "STBOX ((1.0, 2.0), (1.0, 2.0))",
>>> "STBOX Z((1.0, 2.0, 3.0), (1.0, 2.0, 3.0))",
>>> "STBOX T((1.0, 2.0, 2001-01-03 00:00:00+01), (1.0, 2.0, 2001-01-03␣
↪00:00:00+01))",
>>> "STBOX ZT((1.0, 2.0, 3.0, 2001-01-04 00:00:00+01), (1.0, 2.0, 3.0, 2001-01-04␣
↪00:00:00+01))",
>>> "STBOX T(, 2001-01-03 00:00:00+01), (, 2001-01-03 00:00:00+01))",
>>> "GEODSTBOX((1.0, 2.0, 3.0), (1.0, 2.0, 3.0))",
>>> "GEODSTBOX T((1.0, 2.0, 3.0, 2001-01-03 00:00:00+01), (1.0, 2.0, 3.0, 2001-01-
↪04 00:00:00+01))",
>>> "GEODSTBOX T((, 2001-01-03 00:00:00+01), (, 2001-01-03 00:00:00+01))",
>>> "SRID=5676;STBOX T((1.0, 2.0, 2001-01-04), (1.0, 2.0, 2001-01-04))",
>>> "SRID=4326;GEODSTBOX((1.0, 2.0, 3.0), (1.0, 2.0, 3.0))",
```

Another possibility is to give the bounds in the following order: `xmin`, `ymin`, `zmin`, `tmin`, `xmax`, `ymax`, `zmax`, `tmax`, where the bounds can be instances of `str`, `float` and `datetime`. All arguments are optional but they must be given in pairs for each dimension and at least one pair must be given. When three pairs are given, by default, the third pair will be interpreted as representing the `Z` dimension unless the `dimt` parameter is given. Finally, the `geodetic` parameter determines whether the coordinates in the bounds are planar or spherical.

```
>>> STBox((1.0, 2.0, 1.0, 2.0))
>>> STBox((1.0, 2.0, 3.0, 1.0, 2.0, 3.0))
>>> STBox((1.0, 2.0, '2001-01-03', 1.0, 2.0, '2001-01-03'), dimt=True)
>>> STBox((1.0, 2.0, 3.0, '2001-01-04', 1.0, 2.0, 3.0, '2001-01-04'))
>>> STBox(('2001-01-03', '2001-01-03'))
>>> STBox((1.0, 2.0, 3.0, 1.0, 2.0, 3.0), geodetic=True)
>>> STBox((1.0, 2.0, 3.0, '2001-01-04', 1.0, 2.0, 3.0, '2001-01-03'),␣
↪geodetic=True)
>>> STBox((1.0, 2.0, 3.0, '2001-01-04', 1.0, 2.0, 3.0, '2001-01-03'),␣
↪geodetic=True, srid=4326)
>>> STBox(('2001-01-03', '2001-01-03'), geodetic=True)
```

xmin
Minimum X

ymin
Minimum Y

zmin
Minimum Z

tmin
Minimum T

xmax
Maximum X

ymax
Maximum Y

zmax
Maximum Z

tmax
Maximum T

geodetic
Is the box is geodetic?

srid
SRID of the geographic coordinates

3.4 Main Types

class `mobilitydb.main.`**`TBool`**
Bases: `mobilitydb.temporal.temporal.Temporal`

Abstract class for representing temporal Booleans of any duration.

class `mobilitydb.main.`**`TBoolInst`**(*value*, *time=None*)
Bases: `mobilitydb.temporal.temporalinst.TemporalInst`, `mobilitydb.main.tbool.TBool`

Class for representing temporal Booleans of instant duration.

`TBoolInst` objects can be created with a single argument of type string as in MobilityDB.

```
>>> TBoolInst('true@2019-09-01')
```

Another possibility is to give the `value` and the `time` arguments, which can be instances of `str`, `bool`, or `datetime`.

```
>>> TBoolInst('True', '2019-09-08 00:00:00+01')
>>> TBoolInst(['True', '2019-09-08 00:00:00+01'])
>>> TBoolInst(True, parse('2019-09-08 00:00:00+01'))
>>> TBoolInst([True, parse('2019-09-08 00:00:00+01')])
```

class `mobilitydb.main.`**`TBoolI`**(**argv*)
Bases: `mobilitydb.temporal.temporali.TemporalI`, `mobilitydb.main.tbool.TBool`

Class for representing temporal Booleans of instant set duration.

`TBoolI` objects can be created with a single argument of type string as in MobilityDB.

```
>>> TBoolI('AA@2019-09-01')
```

Another possibility is to give a tuple or list of arguments, which can be instances of `str` or `TBoolInst`.

```
>>> TBoolI('AA@2019-09-01 00:00:00+01', 'BB@2019-09-02 00:00:00+01', 'AA@2019-09-
↪03 00:00:00+01')
>>> TBoolI(TBoolInst('AA@2019-09-01 00:00:00+01'), TBoolInst('BB@2019-09-02␣
↪00:00:00+01'), TBoolInst('AA@2019-09-03 00:00:00+01'))
>>> TBoolI(['AA@2019-09-01 00:00:00+01', 'BB@2019-09-02 00:00:00+01', 'AA@2019-09-
↪03 00:00:00+01'])
>>> TBoolI([TBoolInst('AA@2019-09-01 00:00:00+01'), TBoolInst('BB@2019-09-02␣
↪00:00:00+01'), TBoolInst('AA@2019-09-03 00:00:00+01')])
```

class `mobilitydb.main.`**`TBoolSeq`**(*instantList*, *lower_inc=None*, *upper_inc=None*)

Bases: `mobilitydb.temporal.temporalseq.TemporalSeq`, `mobilitydb.main.tbool.TBool`

Class for representing temporal Booleans of sequence duration.

`TBoolSeq` objects can be created with a single argument of type string as in MobilityDB.

```
>>> TBoolSeq('[true@2019-09-01 00:00:00+01, false@2019-09-02 00:00:00+01,␣
↪true@2019-09-03 00:00:00+01]')
```

Another possibility is to give the arguments as follows.

- `instantList` is the list of composing instants, which can be instances of `str` or `TBoolInst`,
- `lower_inc` and `upper_inc` are instances of `bool` specifying whether the bounds are inclusive or not. By default `lower_inc` is `True` and `upper_inc` is `False`.

Some examples are given next.

```
>>> TBoolSeq(['true@2019-09-01 00:00:00+01', 'false@2019-09-02 00:00:00+01',
↪'true@2019-09-03 00:00:00+01'])
>>> TBoolSeq(TBoolInst('true@2019-09-01 00:00:00+01'), TBoolInst('false@2019-09-
↪02 00:00:00+01'), TBoolInst('true@2019-09-03 00:00:00+01')])
>>> TBoolSeq(['true@2019-09-01 00:00:00+01', 'false@2019-09-02 00:00:00+01',
↪'true@2019-09-03 00:00:00+01'], True, True)
>>> TBoolSeq([TBoolInst('true@2019-09-01 00:00:00+01'), TBoolInst('false@2019-09-
↪02 00:00:00+01'), TBoolInst('true@2019-09-03 00:00:00+01')], True, True)
```

classmethod interpolation()

Interpolation of the temporal value, that is, `'Stepwise'`.

class `mobilitydb.main.`**`TBoolS`**(*sequenceList*)

Bases: `mobilitydb.temporal.temporals.TemporalS`, `mobilitydb.main.tbool.TBool`

Class for representing temporal Booleans of sequence set duration.

`TBoolS` objects can be created with a single argument of type string as in MobilityDB.

```
>>> TBoolS('{[true@2019-09-01 00:00:00+01], [false@2019-09-02 00:00:00+01,␣
↪true@2019-09-03 00:00:00+01]}')
```

Another possibility is to give the list of composing sequences, which can be instances of `str` or `TBoolSeq`.

```
>>> TBoolS(['[true@2019-09-01 00:00:00+01]', '[false@2019-09-02 00:00:00+01,␣
↪true@2019-09-03 00:00:00+01]'])
>>> TBoolS([TBoolSeq('[true@2019-09-01 00:00:00+01]'), TBoolSeq('[false@2019-09-
↪02 00:00:00+01, true@2019-09-03 00:00:00+01]')])
>>> TBoolS([TBoolSeq('[true@2019-09-01 00:00:00+01]'), TBoolSeq('[false@2019-09-
↪02 00:00:00+01, true@2019-09-03 00:00:00+01]')])
```

classmethod interpolation()
Interpolation of the temporal value, that is, `'Stepwise'`.

class mobilitydb.main.**TInt**
Bases: `mobilitydb.temporal.temporal.Temporal`

Abstract class for representing temporal integers of any duration.

valueRange
Range of values taken by the temporal value as defined by its minimum and maximum value

class mobilitydb.main.**TIntInst**(*value*, *time=None*)
Bases: `mobilitydb.temporal.temporalinst.TemporalInst`, `mobilitydb.main.tint.TInt`

Class for representing temporal integers of instant duration.

`TIntInst` objects can be created with a single argument of type string as in MobilityDB.

```
>>> TIntInst('10@2019-09-01')
```

Another possibility is to give the `value` and the `time` arguments, which can be instances of `str`, `int` or `datetime`.

```
>>> TIntInst('10', '2019-09-08 00:00:00+01')
>>> TIntInst(['10', '2019-09-08 00:00:00+01'])
>>> TIntInst(10, parse('2019-09-08 00:00:00+01'))
>>> TIntInst([10, parse('2019-09-08 00:00:00+01')])
```

class mobilitydb.main.**TIntI**(**argv*)
Bases: `mobilitydb.temporal.temporali.TemporalI`, `mobilitydb.main.tint.TInt`

Class for representing temporal integers of instant set duration.

`TIntI` objects can be created with a single argument of type string as in MobilityDB.

```
>>> TIntI('10@2019-09-01')
```

Another possibility is to give a tuple or list of composing instants, which can be instances of `str` or `TIntInst`.

```
>>> TIntI('10@2019-09-01 00:00:00+01', '20@2019-09-02 00:00:00+01', '10@2019-09-
↪03 00:00:00+01')
>>> TIntI(TIntInst('10@2019-09-01 00:00:00+01'), TIntInst('20@2019-09-02␣
↪00:00:00+01'), TIntInst('10@2019-09-03 00:00:00+01'))
>>> TIntI(['10@2019-09-01 00:00:00+01', '20@2019-09-02 00:00:00+01', '10@2019-09-
↪03 00:00:00+01'])
>>> TIntI([TIntInst('10@2019-09-01 00:00:00+01'), TIntInst('20@2019-09-02␣
↪00:00:00+01'), TIntInst('10@2019-09-03 00:00:00+01')])
```

class mobilitydb.main.**TIntSeq**(*instantList*, *lower_inc=None*, *upper_inc=None*)
Bases: `mobilitydb.temporal.temporalseq.TemporalSeq`, `mobilitydb.main.tint.TInt`

Class for representing temporal integers of sequence duration.

`TIntSeq` objects can be created with a single argument of type string as in MobilityDB.

```
>>> TIntSeq('[10@2019-09-01 00:00:00+01, 20@2019-09-02 00:00:00+01, 10@2019-09-03␣
↪00:00:00+01]')
```

Another possibility is to give the arguments as follows:

- `instantList` is the list of composing instants, which can be instances of `str` or `TIntInst`,

- `lower_inc` and `upper_inc` are instances of `bool` specifying whether the bounds are inclusive or not. By default `lower_inc` is `True` and `upper_inc` is `False`.

Some examples are given next.

```
>>> TIntSeq(['10@2019-09-01 00:00:00+01', '20@2019-09-02 00:00:00+01', '10@2019-
↪09-03 00:00:00+01'])
>>> TIntSeq([TIntInst('10@2019-09-01 00:00:00+01'), TIntInst('20@2019-09-02␣
↪00:00:00+01'), TIntInst('10@2019-09-03 00:00:00+01')])
>>> TIntSeq(['10@2019-09-01 00:00:00+01', '20@2019-09-02 00:00:00+01', '10@2019-
↪09-03 00:00:00+01'], True, True)
>>> TIntSeq([TIntInst('10@2019-09-01 00:00:00+01'), TIntInst('20@2019-09-02␣
↪00:00:00+01'), TIntInst('10@2019-09-03 00:00:00+01')], True, True)
```

classmethod interpolation()

Interpolation of the temporal value, that is, `'Stepwise'`.

class mobilitydb.main.**TIntS**(*sequenceList*)

Bases: `mobilitydb.temporal.temporals.TemporalS`, `mobilitydb.main.tint.TInt`

Class for representing temporal integers of sequence duration.

`TIntS` objects can be created with a single argument of type string as in MobilityDB.

```
>>> TIntS('{[10@2019-09-01 00:00:00+01], [20@2019-09-02 00:00:00+01, 10@2019-09-
↪03 00:00:00+01]}')
```

Another possibility is to give the list of composing sequences, which can be instances of `str` or `TIntSeq`.

```
>>> TIntS(['[10@2019-09-01 00:00:00+01]', '[20@2019-09-02 00:00:00+01, 10@2019-09-
↪03 00:00:00+01]'])
>>> TIntS([TIntSeq('[10@2019-09-01 00:00:00+01]'), TIntSeq('[20@2019-09-02␣
↪00:00:00+01, 10@2019-09-03 00:00:00+01]')])
>>> TIntS([TIntSeq('[10@2019-09-01 00:00:00+01]'), TIntSeq('[20@2019-09-02␣
↪00:00:00+01, 10@2019-09-03 00:00:00+01]')])
```

classmethod interpolation()

Interpolation of the temporal value, that is, `'Stepwise'`.

class mobilitydb.main.**TFloat**

Bases: `mobilitydb.temporal.temporal.Temporal`

Abstract class for representing temporal floats of any duration.

valueRange

Range of values taken by the temporal value as defined by its minimum and maximum value

class mobilitydb.main.**TFloatInst**(*value*, *time=None*)

Bases: `mobilitydb.temporal.temporalinst.TemporalInst`, `mobilitydb.main.tfloat.TFloat`

Class for representing temporal floats of instant duration.

`TFloatInst` objects can be created with a single argument of type string as in MobilityDB.

```
>>> TFloatInst('10.0@2019-09-01')
```

Another possibility is to give the `value` and the `time` arguments, which can be instances of `str`, `float` or `datetime`.

```
>>> TFloatInst('10.0', '2019-09-08 00:00:00+01')
>>> TFloatInst(['10.0', '2019-09-08 00:00:00+01'])
>>> TFloatInst(10.0, parse('2019-09-08 00:00:00+01'))
>>> TFloatInst([10.0, parse('2019-09-08 00:00:00+01')])
```

getValues
List of ranges representing the values taken by the temporal value

class mobilitydb.main.**TFloatI**(**argv*)
Bases: `mobilitydb.temporal.temporali.TemporalI`, `mobilitydb.main.tfloat.TFloat`

Class for representing temporal floats of instant set duration.

`TFloatI` objects can be created with a single argument of type string as in MobilityDB.

```
>>> TFloatI('10.0@2019-09-01')
```

Another possibility is to give a tuple or list of composing instants, which can be instances of `str` or `TFloatInst`.

```
>>> TFloatI('10.0@2019-09-01 00:00:00+01', '20.0@2019-09-02 00:00:00+01', '10.
↪0@2019-09-03 00:00:00+01')
>>> TFloatI(TFloatInst('10.0@2019-09-01 00:00:00+01'), TFloatInst('20.0@2019-09-
↪02 00:00:00+01'), TFloatInst('10.0@2019-09-03 00:00:00+01'))
>>> TFloatI(['10.0@2019-09-01 00:00:00+01', '20.0@2019-09-02 00:00:00+01', '10.
↪0@2019-09-03 00:00:00+01'])
>>> TFloatI([TFloatInst('10.0@2019-09-01 00:00:00+01'), TFloatInst('20.0@2019-09-
↪02 00:00:00+01'), TFloatInst('10.0@2019-09-03 00:00:00+01')])
```

getValues
List of ranges representing the values taken by the temporal value.

class mobilitydb.main.**TFloatSeq**(*instantList*, *lower_inc=None*, *upper_inc=None*, *interp=None*)
Bases: `mobilitydb.temporal.temporalseq.TemporalSeq`, `mobilitydb.main.tfloat.TFloat`

Class for representing temporal floats of sequence duration.

`TFloatSeq` objects can be created with a single argument of type string as in MobilityDB.

```
>>> TFloatSeq('[10.0@2019-09-01 00:00:00+01, 20.0@2019-09-02 00:00:00+01, 10.
↪0@2019-09-03 00:00:00+01]')
>>> TFloatSeq('Interp=Stepwise;[10.0@2019-09-01 00:00:00+01, 20.0@2019-09-02␣
↪00:00:00+01, 10.0@2019-09-03 00:00:00+01]')
```

Another possibility is to give the arguments as follows:

- `instantList` is the list of composing instants, which can be instances of `str` or `TFloatInst`,
- `lower_inc` and `upper_inc` are instances of `bool` specifying whether the bounds are inclusive or not. By default `lower_inc` is `True` and `upper_inc` is `False`.
- `interp` which is either `'Linear'` or `'Stepwise'`, the former being the default.

Some examples are shown next.

```
>>> TFloatSeq(['10.0@2019-09-01 00:00:00+01', '20.0@2019-09-02 00:00:00+01', '10.
↪0@2019-09-03 00:00:00+01'])
>>> TFloatSeq([TFloatInst('10.0@2019-09-01 00:00:00+01'), TFloatInst('20.0@2019-
↪09-02 00:00:00+01'), TFloatInst('10.0@2019-09-03 00:00:00+01')])
```

(continues on next page)

(continued from previous page)

```
>>> TFloatSeq(['10.0@2019-09-01 00:00:00+01', '20.0@2019-09-02 00:00:00+01', '10.
↪0@2019-09-03 00:00:00+01'], True, True, 'Stepwise')
>>> TFloatSeq([TFloatInst('10.0@2019-09-01 00:00:00+01'), TFloatInst('20.0@2019-
↪09-02 00:00:00+01'), TFloatInst('10.0@2019-09-03 00:00:00+01')], True, True,
↪'Stepwise')
```

interpolation
Interpolation of the temporal value, which is either `'Linear'` or `'Stepwise'`.

getValues
List of ranges representing the values taken by the temporal value.

class `mobilitydb.main.`**`TFloatS`**(*sequenceList*, *interp=None*)
Bases: `mobilitydb.temporal.temporals.TemporalS`, `mobilitydb.main.tfloat.TFloat`

Class for representing temporal floats of sequence duration.

`TFloatS` objects can be created with a single argument of type string as in MobilityDB.

```
>>> TFloatS('{[10.0@2019-09-01 00:00:00+01], [20.0@2019-09-02 00:00:00+01, 10.
↪0@2019-09-03 00:00:00+01]}')
>>> TFloatS('Interp=Stepwise;{[10.0@2019-09-01 00:00:00+01], [20.0@2019-09-02␣
↪00:00:00+01, 10.0@2019-09-03 00:00:00+01]}')
```

Another possibility is to give the arguments as follows:

- `sequenceList` is a list of composing sequences, which can be instances of `str` or `TFloatSeq`,
- `interp` can be `'Linear'` or `'Stepwise'`, the former being the default.

Some examples are shown next.

```
>>> TFloatS(['[10.0@2019-09-01 00:00:00+01]', '[20.0@2019-09-02 00:00:00+01, 10.
↪0@2019-09-03 00:00:00+01]'])
>>> TFloatS(['[10.0@2019-09-01 00:00:00+01]', '[20.0@2019-09-02 00:00:00+01, 10.
↪0@2019-09-03 00:00:00+01]'], 'Linear')
>>> TFloatS(['Interp=Stepwise;[10.0@2019-09-01 00:00:00+01]', 'Interp=Stepwise;
↪[20.0@2019-09-02 00:00:00+01, 10.0@2019-09-03 00:00:00+01]'], 'Stepwise')
>>> TFloatS([TFloatSeq('[10.0@2019-09-01 00:00:00+01]'), TFloatSeq('[20.0@2019-09-
↪02 00:00:00+01, 10.0@2019-09-03 00:00:00+01]')])
>>> TFloatS([TFloatSeq('[10.0@2019-09-01 00:00:00+01]'),  TFloatSeq('[20.0@2019-
↪09-02 00:00:00+01, 10.0@2019-09-03 00:00:00+01]')], 'Linear')
>>> TFloatS([TFloatSeq('Interp=Stepwise;[10.0@2019-09-01 00:00:00+01]'),␣
↪TFloatSeq('Interp=Stepwise;[20.0@2019-09-02 00:00:00+01, 10.0@2019-09-03␣
↪00:00:00+01]')], 'Stepwise')
```

interpolation
Interpolation of the temporal value, which is either `'Linear'` or `'Stepwise'`.

getValues
List of ranges representing the values taken by the temporal value

class `mobilitydb.main.`**`TText`**
Bases: `mobilitydb.temporal.temporal.Temporal`

Abstract class for representing temporal strings of any duration.

class `mobilitydb.main.`**`TTextInst`**(*value*, *time=None*)
Bases: `mobilitydb.temporal.temporalinst.TemporalInst`, `mobilitydb.main.ttext.TText`

Class for representing temporal strings of instant duration.

`TTextInst` objects can be created with a single argument of type string as in MobilityDB.

```
>>> TTextInst('AA@2019-09-01')
```

Another possibility is to give the `value` and the `time` arguments, which can be instances of `str` or `datetime`.

```
>>> TTextInst('AA', '2019-09-08 00:00:00+01')
>>> TTextInst(['AA', '2019-09-08 00:00:00+01'])
>>> TTextInst('AA', parse('2019-09-08 00:00:00+01'))
>>> TTextInst(['AA', parse('2019-09-08 00:00:00+01')])
```

class `mobilitydb.main.`**`TTextI`**(**argv*)

Bases: `mobilitydb.temporal.temporali.TemporalI`, `mobilitydb.main.ttext.TText`

Class for representing temporal strings of instant set duration.

`TTextI` objects can be created with a single argument of type string as in MobilityDB.

```
>>> TTextI('AA@2019-09-01')
```

Another possibility is to give a tuple or list of composing instants, which can be instances of `str` or `TTextInst`.

```
>>> TTextI('AA@2019-09-01 00:00:00+01', 'BB@2019-09-02 00:00:00+01', 'AA@2019-09-
↪03 00:00:00+01')
>>> TTextI(TTextInst('AA@2019-09-01 00:00:00+01'), TTextInst('BB@2019-09-02␣
↪00:00:00+01'), TTextInst('AA@2019-09-03 00:00:00+01'))
>>> TTextI(['AA@2019-09-01 00:00:00+01', 'BB@2019-09-02 00:00:00+01', 'AA@2019-09-
↪03 00:00:00+01'])
>>> TTextI([TTextInst('AA@2019-09-01 00:00:00+01'), TTextInst('BB@2019-09-02␣
↪00:00:00+01'), TTextInst('AA@2019-09-03 00:00:00+01')])
```

class `mobilitydb.main.`**`TTextSeq`**(*instantList*, *lower_inc=None*, *upper_inc=None*)

Bases: `mobilitydb.temporal.temporalseq.TemporalSeq`, `mobilitydb.main.ttext.TText`

Class for representing temporal strings of sequence duration.

`TTextSeq` objects can be created with a single argument of type string as in MobilityDB.

```
>>> TTextSeq('[AA@2019-09-01 00:00:00+01, BB@2019-09-02 00:00:00+01, AA@2019-09-
↪03 00:00:00+01]')
```

Another possibility is to give the arguments as follows:

- `instantList` is the list of composing instants, which can be instances of `str` or `TTextInst`,
- `lower_inc` and `upper_inc` are instances of `bool` specifying whether the bounds are inclusive or not. By default `lower_inc` is `True` and `upper_inc` is `False`.

Some examples are given next.

```
>>> TTextSeq(['AA@2019-09-01 00:00:00+01', 'BB@2019-09-02 00:00:00+01', 'AA@2019-
↪09-03 00:00:00+01'])
>>> TTextSeq(TTextInst('AA@2019-09-01 00:00:00+01'), TTextInst('BB@2019-09-02␣
↪00:00:00+01'), TTextInst('AA@2019-09-03 00:00:00+01')])
```

(continues on next page)

(continued from previous page)

```
>>> TTextSeq(['AA@2019-09-01 00:00:00+01', 'BB@2019-09-02 00:00:00+01', 'AA@2019-
↪09-03 00:00:00+01'], True, True)
>>> TTextSeq([TTextInst('AA@2019-09-01 00:00:00+01'), TTextInst('BB@2019-09-02␣
↪00:00:00+01'), TTextInst('AA@2019-09-03 00:00:00+01')], True, True)
```

classmethod interpolation()
Interpolation of the temporal value, that is, `'Stepwise'`.

class mobilitydb.main.**TTextS**(*sequenceList*)
Bases: `mobilitydb.temporal.temporals.TemporalS`, `mobilitydb.main.ttext.TText`

Class for representing temporal strings of sequence duration.

`TTextS` objects can be created with a single argument of typestring as in MobilityDB.

```
>>> TTextS('{[AA@2019-09-01 00:00:00+01], [BB@2019-09-02 00:00:00+01, AA@2019-09-
↪03 00:00:00+01]}')
```

Another possibility is to give the list of composing sequences, which can be instances of `str` or `TTextSeq`.

```
>>> TTextS(['[AA@2019-09-01 00:00:00+01]', '[BB@2019-09-02 00:00:00+01, AA@2019-
↪09-03 00:00:00+01]'])
>>> TTextS([TTextSeq('[AA@2019-09-01 00:00:00+01]'), TTextSeq('[BB@2019-09-02␣
↪00:00:00+01, AA@2019-09-03 00:00:00+01]')])
>>> TTextS([TTextSeq('[AA@2019-09-01 00:00:00+01]'), TTextSeq('[BB@2019-09-02␣
↪00:00:00+01, AA@2019-09-03 00:00:00+01]')])
```

classmethod interpolation()
Interpolation of the temporal value, that is, `'Stepwise'`.

class mobilitydb.main.**TPointInst**(*value*, *time=None*, *srid=None*)
Bases: `mobilitydb.temporal.temporalinst.TemporalInst`

Abstract class for representing temporal points of instant duration.

getValues
Geometry representing the values taken by the temporal value.

class mobilitydb.main.**TPointI**(**argv*, *srid=None*)
Bases: `mobilitydb.temporal.temporali.TemporalI`

Abstract class for representing temporal points of instant set duration.

getValues
Geometry representing the values taken by the temporal value.

class mobilitydb.main.**TPointSeq**(*instantList*, *lower_inc=None*, *upper_inc=None*, *interp=None*, *srid=None*)
Bases: `mobilitydb.temporal.temporalseq.TemporalSeq`

Abstract class for representing temporal points of sequence duration.

interpolation
Interpolation of the temporal value, which is either `'Linear'` or `'Stepwise'`.

getValues
Geometry representing the values taken by the temporal value.

class mobilitydb.main.**TPointS**(*sequenceList*, *interp=None*, *srid=None*)
Bases: `mobilitydb.temporal.temporals.TemporalS`

Abstract class for representing temporal points of sequence set duration.

interpolation
Interpolation of the temporal value, which is either `'Linear'` or `'Stepwise'`.

getValues
Geometry representing the values taken by the temporal value.

class `mobilitydb.main.`**TGeomPoint**
Bases: `mobilitydb.temporal.temporal.Temporal`

Abstract class for representing temporal geometric or geographic points of any duration.

hasz
Does the temporal point has Z dimension?

srid
Returns the SRID.

class `mobilitydb.main.`**TGeomPointInst**(*value*, *time=None*, *srid=None*)
Bases: `mobilitydb.main.tpoint.TPointInst`, `mobilitydb.main.tpoint.TGeomPoint`

Class for representing temporal geometric points of instant duration.

`TGeomPointInst` objects can be created with a single argument of type string as in MobilityDB.

```
>>> TGeomPointInst('Point(10.0 10.0)@2019-09-01')
>>> TGeomPointInst('SRID=4326,Point(10.0 10.0)@2019-09-01')
```

Another possibility is to give the `value` and the `time` arguments, which can be instances of `str`, `Point` or `datetime`. Additionally, the SRID can be specified, it will be 0 by default if not given.

```
>>> TGeomPointInst('Point(10.0 10.0)', '2019-09-08 00:00:00+01', 4326)
>>> TGeomPointInst(['Point(10.0 10.0)', '2019-09-08 00:00:00+01', 4326])
>>> TGeomPointInst(Point(10.0, 10.0), parse('2019-09-08 00:00:00+01'), 4326)
>>> TGeomPointInst([Point(10.0, 10.0), parse('2019-09-08 00:00:00+01'), 4326])
```

class `mobilitydb.main.`**TGeomPointI**(**argv*, ***kwargs*)
Bases: `mobilitydb.main.tpoint.TPointI`, `mobilitydb.main.tpoint.TGeomPoint`

Class for representing temporal geometric points of instant set duration.

`TGeomPointI` objects can be created with a single argument of type string as in MobilityDB.

```
>>> TGeomPointI('Point(10.0 10.0)@2019-09-01')
```

Another possibility is to give a tuple or list of arguments specifying the composing instants, which can be instances of `str` or `TGeomPointInst`.

```
>>> TGeomPointI('Point(10.0 10.0)@2019-09-01 00:00:00+01', 'Point(20.0 20.0)@2019-
↪09-02 00:00:00+01', 'Point(10.0 10.0)@2019-09-03 00:00:00+01')
>>> TGeomPointI(TGeomPointInst('Point(10.0 10.0)@2019-09-01 00:00:00+01'),␣
↪TGeomPointInst('Point(20.0 20.0)@2019-09-02 00:00:00+01'), TGeomPointInst(
↪'Point(10.0 10.0)@2019-09-03 00:00:00+01'))
>>> TGeomPointI(['Point(10.0 10.0)@2019-09-01 00:00:00+01', 'Point(20.0 20.
↪0)@2019-09-02 00:00:00+01', 'Point(10.0 10.0)@2019-09-03 00:00:00+01'])
>>> TGeomPointI([TGeomPointInst('Point(10.0 10.0)@2019-09-01 00:00:00+01'),␣
↪TGeomPointInst('Point(20.0 20.0)@2019-09-02 00:00:00+01'), TGeomPointInst(
↪'Point(10.0 10.0)@2019-09-03 00:00:00+01')])
```

class `mobilitydb.main.`**TGeomPointSeq**(*instantList*, *lower_inc=None*, *upper_inc=None*, *interp=None*, *srid=None*)
Bases: `mobilitydb.main.tpoint.TPointSeq`, `mobilitydb.main.tpoint.TGeomPoint`

Class for representing temporal geometric points of sequence duration.

`TGeomPointSeq` objects can be created with a single argument of type string as in MobilityDB.

```
>>> TGeomPointSeq('[Point(10.0 10.0)@2019-09-01 00:00:00+01, Point(20.0 20.
↪0)@2019-09-02 00:00:00+01, Point(10.0 10.0)@2019-09-03 00:00:00+01]')
>>> TGeomPointSeq('Interp=Stepwise;[Point(10.0 10.0)@2019-09-01 00:00:00+01,␣
↪Point(20.0 20.0)@2019-09-02 00:00:00+01, Point(10.0 10.0)@2019-09-03␣
↪00:00:00+01]')
```

Another possibility is to give the arguments as follows:

- `instantList` is the list of composing instants, which can be instances of `str` or `TGeogPointInst`,
- `lower_inc` and `upper_inc` are instances of `bool` specifying whether the bounds are inclusive or not, where by default '*lower_inc*' is `True` and `upper_inc` is `False`,
- `interp` which is either `'Linear'` or `'Stepwise'`, the former being the default, and
- `srid` is an integer specifiying the SRID

Some examples are shown next.

```
>>> TGeomPointSeq(['Point(10.0 10.0)@2019-09-01 00:00:00+01', 'Point(20.0 20.
↪0)@2019-09-02 00:00:00+01', 'Point(10.0 10.0)@2019-09-03 00:00:00+01'])
>>> TGeomPointSeq([TGeomPointInst('Point(10.0 10.0)@2019-09-01 00:00:00+01'),␣
↪TGeomPointInst('Point(20.0 20.0)@2019-09-02 00:00:00+01'), TGeomPointInst(
↪'Point(10.0 10.0)@2019-09-03 00:00:00+01')])
>>> TGeomPointSeq(['Point(10.0 10.0)@2019-09-01 00:00:00+01', 'Point(20.0 20.
↪0)@2019-09-02 00:00:00+01', 'Point(10.0 10.0)@2019-09-03 00:00:00+01'], True,␣
↪True, 'Stepwise')
>>> TGeomPointSeq([TGeomPointInst('Point(10.0 10.0)@2019-09-01 00:00:00+01'),␣
↪TGeomPointInst('Point(20.0 20.0)@2019-09-02 00:00:00+01'), TGeomPointInst(
↪'Point(10.0 10.0)@2019-09-03 00:00:00+01')], True, True, 'Stepwise')
```

class `mobilitydb.main.`**`TGeomPointS`**(*sequenceList*, *interp=None*, *srid=None*)

Bases: `mobilitydb.main.tpoint.TPointS`, `mobilitydb.main.tpoint.TGeomPoint`

Class for representing temporal geometric points of sequence duration.

`TGeomPointS` objects can be created with a single argument of type string as in MobilityDB.

```
>>> TGeomPointS('{[Point(10.0 10.0)@2019-09-01 00:00:00+01], [Point(20.0 20.
↪0)@2019-09-02 00:00:00+01, Point(10.0 10.0)@2019-09-03 00:00:00+01]}')
>>> TGeomPointS('Interp=Stepwise;{[Point(10.0 10.0)@2019-09-01 00:00:00+01],␣
↪[Point(20.0 20.0)@2019-09-02 00:00:00+01, Point(10.0 10.0)@2019-09-03␣
↪00:00:00+01]}')
```

Another possibility is to give the arguments as follows:

- `sequenceList` is the list of composing sequences, which can be instances of `str` or `TGeomPointSeq`,
- `interp` can be `'Linear'` or `'Stepwise'`, the former being the default, and
- `srid` is an integer specifiying the SRID, if will be 0 by default if not given.

Some examples are shown next.

```
>>> TGeomPointS(['[Point(10.0 10.0)@2019-09-01 00:00:00+01]', '[Point(20.0 20.
↪0)@2019-09-02 00:00:00+01, Point(10.0 10.0)@2019-09-03 00:00:00+01]'])
>>> TGeomPointS(['[Point(10.0 10.0)@2019-09-01 00:00:00+01]', '[Point(20.0 20.
↪0)@2019-09-02 00:00:00+01, Point(10.0 10.0)@2019-09-03 00:00:00+01]'], 'Linear')
```

(continues on next page)

(continued from previous page)

```
>>> TGeomPointS(['Interp=Stepwise;[Point(10.0 10.0)@2019-09-01 00:00:00+01]',
↪'Interp=Stepwise;[Point(20.0 20.0)@2019-09-02 00:00:00+01, Point(10.0 10.
↪0)@2019-09-03 00:00:00+01]'], 'Stepwise')
>>> TGeomPointS([TGeomPointSeq('[Point(10.0 10.0)@2019-09-01 00:00:00+01]'),␣
↪TGeomPointSeq('[Point(20.0 20.0)@2019-09-02 00:00:00+01, Point(10.0 10.0)@2019-
↪09-03 00:00:00+01]')])
>>> TGeomPointS([TGeomPointSeq('[Point(10.0 10.0)@2019-09-01 00:00:00+01]'), ␣
↪TGeomPointSeq('[Point(20.0 20.0)@2019-09-02 00:00:00+01, Point(10.0 10.0)@2019-
↪09-03 00:00:00+01]')], 'Linear')
>>> TGeomPointS([TGeomPointSeq('Interp=Stepwise;[Point(10.0 10.0)@2019-09-01␣
↪00:00:00+01]'), TGeomPointSeq('Interp=Stepwise;[Point(20.0 20.0)@2019-09-02␣
↪00:00:00+01, Point(10.0 10.0)@2019-09-03 00:00:00+01]')], 'Stepwise')
```

class `mobilitydb.main.`**`TGeogPoint`**

Bases: `mobilitydb.temporal.temporal.Temporal`

Abstract class for representing temporal geographic points of any duration.

hasz

Does the temporal point has Z dimension?

srid

Returns the SRID.

class `mobilitydb.main.`**`TGeogPointInst`**(*value*, *time=None*, *srid=None*)

Bases: `mobilitydb.main.tpoint.TPointInst`, `mobilitydb.main.tpoint.TGeogPoint`

Class for representing temporal geographic points of instant duration.

`TGeogPointInst` objects can be created with a single argument of type string as in MobilityDB.

```
>>> TGeogPointInst('Point(10.0 10.0)@2019-09-01')
```

Another possibility is to give the `value` and the `time` arguments, which can be instances of `str`, `Point` or `datetime`. Additionally, the SRID can be specified, it will be 0 by default if not given.

```
>>> TGeogPointInst('Point(10.0 10.0)', '2019-09-08 00:00:00+01')
>>> TGeogPointInst(['Point(10.0 10.0)', '2019-09-08 00:00:00+01'])
>>> TGeogPointInst(Point(10.0, 10.0), parse('2019-09-08 00:00:00+01'))
>>> TGeogPointInst([Point(10.0, 10.0), parse('2019-09-08 00:00:00+01')])
```

class `mobilitydb.main.`**`TGeogPointI`**(**argv*, ***kwargs*)

Bases: `mobilitydb.main.tpoint.TPointI`, `mobilitydb.main.tpoint.TGeogPoint`

Class for representing temporal geometric points of instant set duration.

`TGeogPointI` objects can be created with a single argument of type string as in MobilityDB.

```
>>> TGeogPointI('Point(10.0 10.0)@2019-09-01')
```

Another possibility is to give a tuple or list of arguments specifying the composing instants, which can be instances of `str` or `TGeogPointInst`.

```
>>> TGeogPointI('Point(10.0 10.0)@2019-09-01 00:00:00+01', 'Point(20.0 20.0)@2019-
↪09-02 00:00:00+01', 'Point(10.0 10.0)@2019-09-03 00:00:00+01')
>>> TGeogPointI(TGeogPointInst('Point(10.0 10.0)@2019-09-01 00:00:00+01'),␣
↪TGeogPointInst('Point(20.0 20.0)@2019-09-02 00:00:00+01'), TGeogPointInst(
↪'Point(10.0 10.0)@2019-09-03 00:00:00+01'))
>>> TGeogPointI(['Point(10.0 10.0)@2019-09-01 00:00:00+01', 'Point(20.0 20.
↪0)@2019-09-02 00:00:00+01', 'Point(10.0 10.0)@2019-09-03 00:00:00+01'])
```

(continues on next page)

(continued from previous page)

```
>>> TGeogPointI([TGeogPointInst('Point(10.0 10.0)@2019-09-01 00:00:00+01'),␣
↪TGeogPointInst('Point(20.0 20.0)@2019-09-02 00:00:00+01'), TGeogPointInst(
↪'Point(10.0 10.0)@2019-09-03 00:00:00+01')])
```

class mobilitydb.main.**TGeogPointSeq**(*instantList*, *lower_inc=None*, *upper_inc=None*, *interp=None*, *srid=None*)

Bases: mobilitydb.main.tpoint.TPointSeq, mobilitydb.main.tpoint.TGeogPoint

Class for representing temporal geographic points of sequence duration.

TGeogPointSeq objects can be created with a single argument of type string as in MobilityDB.

```
>>> TGeogPointSeq('[Point(10.0 10.0)@2019-09-01 00:00:00+01, Point(20.0 20.
↪0)@2019-09-02 00:00:00+01, Point(10.0 10.0)@2019-09-03 00:00:00+01]')
>>> TGeogPointSeq('Interp=Stepwise;[Point(10.0 10.0)@2019-09-01 00:00:00+01,␣
↪Point(20.0 20.0)@2019-09-02 00:00:00+01, Point(10.0 10.0)@2019-09-03␣
↪00:00:00+01]')
```

Another possibility is to give the arguments as follows:

- instantList is the list of composing instants, which can be instances of str or TGeogPointInst,
- lower_inc and upper_inc are instances of bool specifying whether the bounds are includive or not, where by default '*lower_inc*' is True and upper_inc is False, and
- interp which is either 'Linear' or 'Stepwise', the former being the default.
- srid is an integer specifiying the SRID

Some examples are shown next.

```
>>> TGeogPointSeq(['Point(10.0 10.0)@2019-09-01 00:00:00+01', 'Point(20.0 20.
↪0)@2019-09-02 00:00:00+01', 'Point(10.0 10.0)@2019-09-03 00:00:00+01'])
>>> TGeogPointSeq([TGeogPointInst('Point(10.0 10.0)@2019-09-01 00:00:00+01'),␣
↪TGeogPointInst('Point(20.0 20.0)@2019-09-02 00:00:00+01'), TGeogPointInst(
↪'Point(10.0 10.0)@2019-09-03 00:00:00+01')])
>>> TGeogPointSeq(['Point(10.0 10.0)@2019-09-01 00:00:00+01', 'Point(20.0 20.
↪0)@2019-09-02 00:00:00+01', 'Point(10.0 10.0)@2019-09-03 00:00:00+01'], True,␣
↪True, 'Stepwise')
>>> TGeogPointSeq([TGeogPointInst('Point(10.0 10.0)@2019-09-01 00:00:00+01'),␣
↪TGeogPointInst('Point(20.0 20.0)@2019-09-02 00:00:00+01'), TGeogPointInst(
↪'Point(10.0 10.0)@2019-09-03 00:00:00+01')], True, True, 'Stepwise')
```

class mobilitydb.main.**TGeogPointS**(*sequenceList*, *interp=None*, *srid=None*)

Bases: mobilitydb.main.tpoint.TPointS, mobilitydb.main.tpoint.TGeogPoint

Class for representing temporal geographic points of sequence duration.

TGeogPointS objects can be created with a single argument of type string as in MobilityDB.

```
>>> TGeogPointS('{[Point(10.0 10.0)@2019-09-01 00:00:00+01], [Point(20.0 20.
↪0)@2019-09-02 00:00:00+01, Point(10.0 10.0)@2019-09-03 00:00:00+01]}')
>>> TGeogPointS('Interp=Stepwise;{[Point(10.0 10.0)@2019-09-01 00:00:00+01],␣
↪[Point(20.0 20.0)@2019-09-02 00:00:00+01, Point(10.0 10.0)@2019-09-03␣
↪00:00:00+01]}')
```

Another possibility is to give the arguments as follows:

- sequenceList is the list of composing sequences, which can be instances of str or TGeogPointSeq,

- `interp` can be `'Linear'` or `'Stepwise'`, the former being the default, and
- `srid` is an integer specifiying the SRID, if will be 0 by default if not given.

Some examples are shown next.

```
>>> TGeogPointS(['[Point(10.0 10.0)@2019-09-01 00:00:00+01]', '[Point(20.0 20.
↪0)@2019-09-02 00:00:00+01, Point(10.0 10.0)@2019-09-03 00:00:00+01]'])
>>> TGeogPointS(['[Point(10.0 10.0)@2019-09-01 00:00:00+01]', '[Point(20.0 20.
↪0)@2019-09-02 00:00:00+01, Point(10.0 10.0)@2019-09-03 00:00:00+01]'], 'Linear')
>>> TGeogPointS(['Interp=Stepwise;[Point(10.0 10.0)@2019-09-01 00:00:00+01]',
↪'Interp=Stepwise;[Point(20.0 20.0)@2019-09-02 00:00:00+01, Point(10.0 10.
↪0)@2019-09-03 00:00:00+01]'], 'Stepwise')
>>> TGeogPointS([TGeogPointSeq('[Point(10.0 10.0)@2019-09-01 00:00:00+01]'),␣
↪TGeogPointSeq('[Point(20.0 20.0)@2019-09-02 00:00:00+01, Point(10.0 10.0)@2019-
↪09-03 00:00:00+01]')])
>>> TGeogPointS([TGeogPointSeq('[Point(10.0 10.0)@2019-09-01 00:00:00+01]'), ␣
↪TGeogPointSeq('[Point(20.0 20.0)@2019-09-02 00:00:00+01, Point(10.0 10.0)@2019-
↪09-03 00:00:00+01]')], 'Linear')
>>> TGeogPointS([TGeogPointSeq('Interp=Stepwise;[Point(10.0 10.0)@2019-09-01␣
↪00:00:00+01]'), TGeogPointSeq('Interp=Stepwise;[Point(20.0 20.0)@2019-09-02␣
↪00:00:00+01, Point(10.0 10.0)@2019-09-03 00:00:00+01]')], 'Stepwise')
```

PYTHON MODULE INDEX

INDEX