Kisters WATER TimeSeries Documentation

Kisters

May 17, 2022

Contents:

1 General API **3**
- 1.1 Time Series . . . 3
- 1.2 Time Series Store . . . 5

2 Stores Backends **9**
- 2.1 File Store . . . 9
- 2.2 KiWIS Store . . . 10
- 2.3 TimeSeriesStoreDecorators . . . 11

3 Examples **13**
- 3.1 KiWIS Example . . . 13

4 Indices and tables **15**

Index **17**

Welcome to the Water Time Series API documentation. This API provides access to time series data from KiWIS and ZRXP files.

CHAPTER 1

General API

1.1 Time Series

TimeSeries is the object representation of the time series data it provides all the attributes and metadata of the time series as well as access to the data itself through the read_data_frame and write_data_frame methods.

class kisters.water.time_series.core.time_series.**TimeSeries**(**args*, ***kwargs*)

Bases: abc.ABC

This class provides the interface of TimeSeries.

coverage_from

The date from which the TimeSeries data starts.

Returns The start date

coverage_until

The date until which the TimeSeries data covers.

Returns The end date

id

The id number which fully identifies the TimeSeries.

Returns The id number

metadata

The map containing all the metadata related to this TimeSeries.

Returns The metadata map

name

The full name of the TimeSeries.

Returns The full name

path

The full path to this TimeSeries.

Returns The path string

read_data_frame(*start: Union[str, datetime.datetime] = None*, *end: Union[str, datetime.datetime] = None*, *params: Mapping = None*, *t0: datetime.datetime = None*, *dispatch_info: str = None*, *member: str = None*, ***kwargs*) → pandas.core.frame.DataFrame

This method returns the TimeSeries data between the start and end dates (both dates included) structured as a pandas DataFrame.

Parameters

- **start** – The starting date from which the data will be returned, expressed either
- **an ISO Datetime string or as a datetime object. If TimeZone is not included,** (*as*) –
- **assumes the TimeZone of the TimeSeries.** (*it*) –
- **end** – The ending date until which the data will be covered (end date included),
- **either as an ISO Datetime string or as a datetime object. If TimeZone** (*expressed*) –
- **not included, it assumes the TimeZone of the TimeSeries.** (*is*) –
- **params** – The parameters passed to the backend call.
- **ensemble time series only** (*For*) –

  To retrieve all data points of all dispatch_infos and members set t0, dispatch_info, member to None including t0, dispatch_info, member as additional columns

  To retrieve a single ensemble member t0: the t0 time stamp of the ensemble member dispatch_info: ensemble dispatch_info identifier member: ensemble member identifier

Returns The DataFrame containing the TimeSeries data

read_ensemble_members(*t0_start: datetime.datetime = None*, *t0_end: datetime.datetime = None*)

Returns a list of dictionaries with the corresponding t0, member and dispatch_infos as key.

Parameters

- **t0_start** – The starting date from which the data will be returned.
- **t0_end** – The ending date until which the data will be covered (end date included).

short_name

The short name of the TimeSeries.

Returns The short name

write_data_frame(*data_frame: pandas.core.frame.DataFrame*, *start: datetime.datetime = None*, *end: datetime.datetime = None*, *t0: datetime.datetime = None*, *dispatch_info: str = None*, *member: str = None*, ***kwargs*)

This methods writes the TimeSeries data from the data_frame into this TimeSeries. If either start or end, cover data missing in the data_frame these date ranges will be deleted. So if you specify an empty DataFrame, you can remove all data between start and end.

Parameters

- **data_frame** – The TimeSeries data to be written in the form of a pandas DataFrame.
- **start** – The date from which data will be writen.
- **end** – The date until which data will be writen (end date included).
- **ensemble time series only** (*For*) –

To write a single ensemble member at once t0: the t0 time stamp of the ensemble member dispatch_info: ensemble dispatch_info identifier member: ensemble member identifier start, end will be ignored

1.2 Time Series Store

TimeSeriesStore provides access to time series data from different backends. All TimeSeriesStore implementations provide this methods, to see details on a concrete Store usage go to *Stores Backends*

class kisters.water.time_series.core.time_series_store.**TimeSeriesStore**

Bases: abc.ABC

create_time_series(*path: str*, *display_name: str*, *attributes: Mapping = None*, *params: Mapping[str, Any] = None*) → kisters.water.time_series.core.time_series.TimeSeries

Create an empty time series.

Parameters

- **path** – The time series path.
- **display_name** – The time series name to display.
- **attributes** – The metadata of the time series.
- **params** – The additional parameters, which are passed to the backend.

create_time_series_from(*copy_from: kisters.water.time_series.core.time_series.TimeSeries*, ***kwargs*) → kisters.water.time_series.core.time_series.TimeSeries

Create a time series as a copy from another existing time series e.g. from another system. This function only copies meta data and only if the underlaying backend supports it.

Parameters **copy_from** – The time series object to be copied.

get_by_filter(*ts_filter: str*, *params: Mapping[str, Any] = None*) → Iterable[kisters.water.time_series.core.time_series.TimeSeries]

Get the time series list by filter.

Parameters

- **ts_filter** – time series path or filter.
- **params** – the additional parameters, which are passed to the backend.

Returns The list of the found TimeSeries objects.

Examples

```
store.get_by_filter("W7AgentTest/20004/S/*")
```

get_by_id(*ts_id: Union[int, str, Iterable[int], Iterable[str]]*, *params: Mapping[str, Any] = None*) → Union[kisters.water.time_series.core.time_series.TimeSeries, Iterable[kisters.water.time_series.core.time_series.TimeSeries]]

Get the time series by id.

Parameters

- **ts_id** – List of ids or single id.
- **params** – The additional parameters, which are passed to the backend.

Returns The time series or a list of time series if ts_id is a list.

Examples

```
store.get_by_id(4711)
store.get_by_id([4711, 4712])
```

get_by_path(*path: str*, *params: Mapping[str, Any] = None*) → kisters.water.time_series.core.time_series.TimeSeries

Get the time series by path.

Parameters

- **path** – The full qualified time series path.
- **params** – The additional parameters, which are passed to the backend.

Returns The TimeSeries object.

Examples

```
store.get_by_path("W7AgentTest/20003/S/cmd")
```

get_entity(*entity_path: str = None*, *entity_id: int = None*, ***kwargs*) → kisters.water.time_series.core.entity.Entity

Retrieve a single Entity from the backend.

Basic implementation is given based on get_entity_list, but it is recommended to overwrite this method based on the backend specifications. :param entity_path: The entity path. :param entity_id: The entity id. :param **kwargs: Additional arguments dependent on the backend.

Returns The matching Entity.

get_entity_list(*entity_filter: str = None*, *entities_id: Iterable[int] = None*, ***kwargs*) → Iterable[kisters.water.time_series.core.entity.Entity]

Retrieve a list of entities from the backend.

The behaviour depends totally on the backend implementation, this should decide if both entity_filter and entities_id arguments can be given and how they behave if both are given. As a rule of thumb if both arguments are given a double filter should be applied. :param entity_filter: The entity filter. :param entities_id: A list of entities id. :param **kwargs: Additional arguments dependent on the backend.

Returns The list of Entity objects.

get_time_series(***kwargs*)

get_time_series_list(***kwargs*)

read_data_frames(*ts_list: Iterable[kisters.water.time_series.core.time_series.TimeSeries], start: Union[str, datetime.datetime] = None, end: Union[str, datetime.datetime] = None, params: Mapping = None, t0: datetime.datetime = None, dispatch_info: str = None, member: str = None*) → Mapping[kisters.water.time_series.core.time_series.TimeSeries, pandas.core.frame.DataFrame]

Read multiple time series as data frames.

Note: TimeSeriesStore provides a default unoptimized implementation. Override this method if your store can provide a more efficient bulk read method.

write_data_frames(*ts_list: Iterable[kisters.water.time_series.core.time_series.TimeSeries], data_frames: Iterable[pandas.core.frame.DataFrame], start: Union[datetime.datetime, Iterable] = None, end: Union[datetime.datetime, Iterable] = None, t0: Union[datetime.datetime, Iterable] = None, dispatch_info: Iterable = None, member: Iterable = None*)

Write multiple data frames to time series

Note: Override this method if your store can provide a better implementation to bulk write data frames.

write_time_series(*ts: kisters.water.time_series.core.time_series.TimeSeries, start: datetime.datetime = None, end: datetime.datetime = None, auto_create: bool = False*) → kisters.water.time_series.core.time_series.TimeSeries

Write a single time series to the time series back end for the given time range.

Parameters

- **ts** – The time series to write.
- **start** – The starting date from which data will be written.
- **end** – The ending date until which data will be written.
- **auto_create** – Create the time series if not exists in the back end.

Examples

```
ts = store1.get_by_id(47122)
store2.write_time_series(ts)
```

write_time_series_list(*ts_list: Iterable[kisters.water.time_series.core.time_series.TimeSeries], start: datetime.datetime = None, end: datetime.datetime = None, auto_create: bool = False*)

Write a time series list to the back end for the given time range.

Parameters

- **ts_list** – The time series list.
- **start** – The starting date from which data will be written.
- **end** – The ending date until which data will be written.
- **auto_create** – Create the time series if not exists in the back end.

Examples

```
ts_list = store1.get_by_filter('W7AgentTest/2000*')
store2.write_time_series_list(ts_list, datetime(2001, 1, 1), datetime(2002, 1,
↪ 1))
```

CHAPTER 2

Stores Backends

2.1 File Store

FileStore class grants access through your own time series data files. Right now, you can have your time series data files in 3 formats, ZRXP, CSV and Pickle.

class kisters.water.time_series.file_io.**FileStore**(*root_dir: str*, *file_format: kisters.water.time_series.file_io.time_series_format.TimeSeri*

FileStore provides a TimeSeriesStore for your local time series data files

Parameters

- **root_dir** – The path to your time series data folder.
- **file_format** – The format used by your time series data files.

Examples

```
from kisters.water.time_series.file_io import FileStore, ZRXPFormat
fs = FileStore('tests/data', ZRXPFormat())
ts = fs.get_by_path('validation/inner_consistency1/station1/H')
```

class kisters.water.time_series.file_io.**CSVFormat**(*delimiter: str = ', ', quotechar: str = '"', header_lines: int = 1*)

CSV formatter class

Example

```
from kisters.water.time_series.file_io import FileStore, CSVFormat
fs = FileStore('tests/data', CSVFormat())
```

class kisters.water.time_series.file_io.**PickleFormat**

Pickle formatter class

Example

```
from kisters.water.time_series.file_io import FileStore, PickleFormat
fs = FileStore('tests/data', PickleFormat())
```

class kisters.water.time_series.file_io.**ZRXPFormat**
ZRXP formatter class

Example

```
from kisters.water.time_series.file_io import FileStore, ZRXPFormat
fs = FileStore('tests/data', ZRXPFormat())
```

2.2 KiWIS Store

KiWISStore class grants access to the time series data of the Kisters Web Interoperability Solution backend.

class kisters.water.time_series.kiwis.**KiWISStore**(*base_url: str*, *datasource: int = 0*, *user: str = None*, *password: str = None*)
Bases: *kisters.water.time_series.core.time_series_store.TimeSeriesStore*

Connector to KiWIS backend over KiWIS REST API

Parameters

- **base_url** – Base url of REST API.
- **data_source** – Optional number identifying the data source.

Examples

```
from kisters.water.time_series.kiwis import KiWISStore
kiwis = KiWISStore('http://kiwis.kisters.de/KiWIS2/KiWIS')
kiwis.get_by_path('DWD/07367/Precip/CmdTotal.1h')
```

_get_time_series(*ts_id: int = None*, *path: str = None*, *params: Mapping[str, Any] = None*) → kisters.water.time_series.core.time_series.TimeSeries
Get a time series identified by id or by path and return a TimeSeries

Parameters

- **ts_id** – The time series id.
- **path** – The time series path (ignored if ts_id is given).
- **params** – The additional parameters, which are passed to the rest api. in addition to the parameters defined by the REST API there are the following keys:

  metadata = comma separated list of additional meta information where the values are “site”, “station”, “parameter”.

  All time series specific information should be specified as is. All station, site and parameter specific information should be prefixed with “station.” etc.

Returns The found TimeSeries object.

Examples

```
kiwis.get_by_id(ts_id = 40765010)

# There is a special key 'all' which allows to retrieve all metadata keys of␣
↪the specified object.
# But this should be handeld with care, because it is expensive.
ts = self.kiwis.get_by_id(ts_id=40765010, params={'metadata': 'station.all'})

# The following statement would retrieve all available information:
kiwis.get_by_id(ts_id=40765010, params={'metadata': 'all,parameter.all,
↪station.all,site.all'})
```

_get_time_series_list(*ts_filter: str = None, id_list: Iterable[int] = None, params: Mapping[str, Any] = None*) → List[kisters.water.time_series.core.time_series.TimeSeries]

Get the time series list and return a list of TimeSeries objects

Parameters

- **ts_filter** – The ts filter.
- **id_list** – The id list.
- **params** – The additional parameters, which are passed to rest api in addition to the parameters defined by the REST API there are the following keys:

  metadata = comma separated list of additional meta information where the values are "site", "station", "parameter".

Returns The list of TimeSeries objects.

Examples

```
kiwis.get_by_filter(ts_filter="FR110031fb-e8e7-4381-a942-372aa8141945/CM0514*
↪")
```

2.3 TimeSeriesStoreDecorators

The TimeSeriesStoreDecorators allow you to "wrap" a TimeSeriesStore to extend its functionality.

class kisters.water.time_series.store_decorators.**AddMetadataStore**(*forward: kisters.water.time_series.core.time_s metadata: Mapping[str, Any]*)

Bases: *kisters.water.time_series.core.time_series_store.TimeSeriesStore*

AddMetadataStore is a TimeSeriesStore decorator which allows to add metadata to TimeSeries inside the original TimeSeriesStore. To add metadata you must provide a Mapping of paths to metadata dictionaries.

Parameters

- **forward** – The TimeSeriesStore to be decorated.

- **metadata** – The mapping providing metadata. Keys are the time series paths, and values are the metadata maps for each time series.

Example

```
import json
from kisters.water.file_io import FileStore, ZRXPFormat
from kisters.water.store_decorators import AddMetadataStore
with open('tests/data/addmetadata.json', 'r') as f:
    j = json.load(f)
store = AddMetadataStore(FileStore('tests/data', ZRXPFormat()), j)
ts = store.get_by_path('validation/threshold/05BJ004.HG.datum.O')
ts.metadata['THRESHOLDATTR']
```

class kisters.water.time_series.store_decorators.**CacheStore**(*forward: kisters.water.time_series.core.time_series_sto*

Bases: *kisters.water.time_series.core.time_series_store.TimeSeriesStore*

CacheStore is a TimeSeriesStore decorator which allows to cache the retrieval of TimeSeries inside the original TimeSeriesStore. Also TimeSeries retrieved this way cache the data they contain in memory.

Parameters **forward** – The TimeSeriesStore to be decorated.

Example

```
from kisters.water.file_io import FileStore, ZRXPFormat
from kisters.water.store_decorators import CacheStore
store = CacheStore(FileStore('tests/data', ZRXPFormat()))
ts = store.get_by_path('validation/threshold/05BJ004.HG.datum.O')
```

CHAPTER 3

Examples

3.1 KiWIS Example

3.1.1 Accessing KiWIS data

This example demonstrates how to access and plot time series data from KiWIS using Python.

```
from datetime import datetime

import matplotlib.pyplot as plt

from kisters.water.time_series.kiwis import KiWISStore

# Initialize KiWISStore object with url to location
kiwis = KiWISStore("http://kiwis.kisters.de/KiWIS2/KiWIS")

# Get time series list by path filter
ts_list = kiwis.get_by_filter("123/*/Precip/MMonth.Total")

# Get time series by path
ts = kiwis.get_by_path("DWD/07367/Precip/CmdTotal.1h")

# We can access timeseries metadata as a dict using the metadata attribute
ts_metadata_dict = ts.metadata
# ts_metadata_dict looks like this:
# {
#     "from": datetime.datetime(
#         2007, 12, 1, 0, 0, tzinfo=tzoffset(None, 3600)
#     ),
#     "to": datetime.datetime(
#         2018, 7, 16, 23, 0, tzinfo=tzoffset(None, 3600)
#     ),
#     "tsPath": "DWD/07367/Precip/CmdTotal.1h",
#     "shortName": "CmdTotal.1h",
```

(continues on next page)

(continued from previous page)

```
#     "id": 7411042,
#     "name": "0 Stundenwerte",
#     "dataCoverageFrom": datetime.datetime(
#         2007, 12, 1, 0, 0, tzinfo=tzoffset(None, 3600)
#     ),
#     "dataCoverageUntil": datetime.datetime(
#         2018, 7, 16, 23, 0, tzinfo=tzoffset(None, 3600)
#     ),
# }

# Access the time series data in the form of pandas DataFrame
df = ts.read_data_frame(datetime(2017, 1, 1), ts.coverage_until)

# Plot the data
plt.figure(figsize=(15, 5))
df["Value"].plot(label=ts.name, figsize=(15, 5))
df["Value"].resample("D").sum().plot(label="Daily sum", figsize=(15, 5))
plt.legend(loc=2)
plt.show()
```

3.1.2 Writing data to KiWIS

Currently KiWIS doesn't allow to write data into or create time series. However, you can use the *File Store* module to save your data locally.

For example, this code snippet continues from the example above and writes the time series variable `ts` to a ZRX file using the ZRXPFormat.

```
from kisters.water.time_series.file_io import ZRXPFormat
writer = ZRXPFormat().writer()
writer.write('my_time_series.zrx', [ts])
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index

Symbols

_get_time_series() (*kisters.water.time_series.kiwis.KiWISStore method*), 10
_get_time_series_list() (*kisters.water.time_series.kiwis.KiWISStore method*), 11

A

AddMetadataStore (*class in kisters.water.time_series.store_decorators*), 11

C

CacheStore (*class in kisters.water.time_series.store_decorators*), 12
coverage_from (*kisters.water.time_series.core.time_series.TimeSeries attribute*), 3
coverage_until (*kisters.water.time_series.core.time_series.TimeSeries attribute*), 3
create_time_series() (*kisters.water.time_series.core.time_series_store.TimeSeriesStore method*), 5
create_time_series_from() (*kisters.water.time_series.core.time_series_store.TimeSeriesStore method*), 5
CSVFormat (*class in kisters.water.time_series.file_io*), 9

F

FileStore (*class in kisters.water.time_series.file_io*), 9

G

get_by_filter() (*kisters.water.time_series.core.time_series_store.TimeSeriesStore method*), 5
get_by_id() (*kisters.water.time_series.core.time_series_store.TimeSeriesStore method*), 5
get_by_path() (*kisters.water.time_series.core.time_series_store.TimeSeriesStore method*), 6
get_entity() (*kisters.water.time_series.core.time_series_store.TimeSeriesStore method*), 6
get_entity_list() (*kisters.water.time_series.core.time_series_store.TimeSeriesStore method*), 6
get_time_series() (*kisters.water.time_series.core.time_series_store.TimeSeriesStore method*), 6
get_time_series_list() (*kisters.water.time_series.core.time_series_store.TimeSeriesStore method*), 6

I

id (*kisters.water.time_series.core.time_series.TimeSeries attribute*), 3

K

KiWISStore (*class in kisters.water.time_series.kiwis*), 10

M

metadata (*kisters.water.time_series.core.time_series.TimeSeries attribute*), 3

N

name (*kisters.water.time_series.core.time_series.TimeSeries attribute*), 3

P

path (*kisters.water.time_series.core.time_series.TimeSeries attribute*), 3
PickleFormat (*class in kisters.water.time_series.file_io*), 9

R

read_time_series() (*kisters.water.time_series.core.time_series.TimeSeries method*), 4

read_data_frames()
(*kisters.water.time_series.core.time_series_store.TimeSeriesStore method*), 6
read_ensemble_members()
(*kisters.water.time_series.core.time_series.TimeSeries method*), 4

S

short_name (*kisters.water.time_series.core.time_series.TimeSeries attribute*), 4

T

TimeSeries (*class in kisters.water.time_series.core.time_series*), 3
TimeSeriesStore (*class in kisters.water.time_series.core.time_series_store*), 5

W

write_data_frame()
(*kisters.water.time_series.core.time_series.TimeSeries method*), 4
write_data_frames()
(*kisters.water.time_series.core.time_series_store.TimeSeriesStore method*), 6
write_time_series()
(*kisters.water.time_series.core.time_series_store.TimeSeriesStore method*), 7
write_time_series_list()
(*kisters.water.time_series.core.time_series_store.TimeSeriesStore method*), 7

Z

ZRXPFormat (*class in kisters.water.time_series.file_io*), 10