HyperStream Documentation

Release 0.3.8

SPHERE WP5

Sep 27, 2019

Contents

HyperStream is a large-scale, flexible and robust software package, written in the Python language, for processing streaming data with workflow creation capabilities. HyperStream overcomes the limitations of other computational engines and provides high-level interfaces to execute complex nesting, fusion, and prediction both in online and offline forms in streaming environments. HyperStream is a general purpose tool that is well-suited for the design, development, and deployment of Machine Learning algorithms and predictive models in a wide space of sequential predictive problems.

The code can be found on our Github project page. It is released under the MIT open source license.

Tutorials:

- Introduction
- Your own tools
- Stream composition
- Real-time streams
- Workflows

Contents:

CHAPTER 1

hyperstream package

1.1 Subpackages

1.1.1 hyperstream.channels package

1.1.1.1 Submodules

1.1.1.2 hyperstream.channels.assets_channel module

Assets channel module.

class hyperstream.channels.assets_channel.**AssetsChannel**(*channel_id*)
Bases: *hyperstream.channels.database_channel.DatabaseChannel*

Assets Channel. Special kind of database channel for static assets and user input data (workflow parameters etc)

create_stream(*stream_id*, *sandbox=None*)
Create the stream

Parameters

- **stream_id** – The stream identifier
- **sandbox** – The sandbox for this stream

Returns None

Raises NotImplementedError

purge_stream(*stream_id*, *remove_definition=False*, *sandbox=None*)
Purge the stream

Parameters

- **stream_id** – The stream identifier
- **remove_definition** – Whether to remove the stream definition as well

- **sandbox** – The sandbox for this stream

Returns None

update_streams(*up_to_timestamp*)
Update the streams

Parameters **up_to_timestamp** –

Returns

write_to_stream(*stream_id*, *data*, *sandbox=None*)
Write to the stream

Parameters

- **stream_id** (StreamId) – The stream identifier
- **data** – The stream instances
- **sandbox** – The sandbox for this stream

Returns None

Raises NotImplementedError

1.1.1.3 hyperstream.channels.assets_channel2 module

1.1.1.4 hyperstream.channels.base_channel module

class hyperstream.channels.base_channel.**BaseChannel**(*channel_id*, *can_calc=False*, *can_create=False*, *calc_agent=None*)
Bases: hyperstream.utils.containers.Printable

Abstract base class for channels

create_stream(*stream_id*, *sandbox=None*)
Must be overridden by deriving classes, must create the stream according to the tool and return its unique identifier stream_id

execute_tool(*stream*, *interval*)
Executes the stream's tool over the given time interval

Parameters

- **stream** – the stream reference
- **interval** – the time interval

Returns None

find_stream(***kwargs*)
Finds a single stream with the given meta data values. Useful for debugging purposes.

Parameters **kwargs** – The meta data as keyword arguments

Returns The stream found

find_streams(***kwargs*)
Finds streams with the given meta data values. Useful for debugging purposes.

Parameters **kwargs** – The meta data as keyword arguments

Returns The streams found

get_or_create_stream(*stream_id*, *try_create=True*)
Helper function to get a stream or create one if it's not already defined

Parameters

- **stream_id** – The stream id
- **try_create** – Whether to try to create the stream if not found

Returns The stream object

get_results(*stream*, *time_interval*)
Must be overridden by deriving classes. 1. Calculates/receives the documents in the stream for the time interval given 2. Returns success or failure and the results (for some channels the values of kwargs can override the return process, e.g. introduce callbacks)

get_stream_writer(*stream*)
Must be overridden by deriving classes, must return a function(document_collection) which writes all the given documents of the form (timestamp,data) from document_collection to the stream Example:

```
.. code-block:: python
```

if stream_id==1:

def f(document_collection):

for (timestamp,data) in document_collection: database[timestamp] = data

return(f)

else: raise Exception('No stream with id '+str(stream_id))

purge_node(*node_id*, *remove_definition=False*, *sandbox=None*)
Purges a node (collection of streams)

Parameters

- **node_id** – The node identifier
- **remove_definition** – Whether to remove the stream definition as well
- **sandbox** – The sandbox

Returns None

purge_stream(*stream_id*, *remove_definition=False*, *sandbox=None*)
Must be overridden by deriving classes, purges the stream and removes the calculated intervals

update_streams(*up_to_timestamp*)
Deriving classes must override this function

1.1.1.5 hyperstream.channels.database_channel module

Database channel module.

class hyperstream.channels.database_channel.**DatabaseChannel**(*channel_id*)
Bases: *hyperstream.channels.base_channel.BaseChannel*

Database Channel. Data stored and retrieved in mongodb using mongoengine.

create_stream(*stream_id*, *sandbox=None*)
Create the stream

Parameters

- **stream_id** – The stream identifier
- **sandbox** – The sandbox for this stream

Returns None

Raises NotImplementedError

get_results(*stream*, *time_interval*)
Get the results for a given stream

Parameters

- **time_interval** – The time interval
- **stream** – The stream object

Returns A generator over stream instances

get_stream_writer(*stream*)
Gets the database channel writer The mongoengine model checks whether a stream_id/datetime pair already exists in the DB (unique pairs) Should be overridden by users' personal channels - allows for non-mongo outputs.

Parameters **stream** – The stream

Returns The stream writer function

purge_stream(*stream_id*, *remove_definition=False*, *sandbox=None*)
Purge the stream

Parameters

- **stream_id** – The stream identifier
- **remove_definition** – Whether to remove the stream definition as well
- **sandbox** – The sandbox for this stream

Returns None

Raises NotImplementedError

update_streams(*up_to_timestamp*)
Update the streams

Parameters **up_to_timestamp** –

Returns

1.1.1.6 hyperstream.channels.file_channel module

class hyperstream.channels.file_channel.**FileChannel**(*channel_id*, *path*, *up_to_timestamp=datetime.datetime(1, 1, 1, 0, 0, tzinfo=<UTC>)*)
Bases: *hyperstream.channels.memory_channel.ReadOnlyMemoryChannel*

An abstract stream channel where the streams are recursive sub-folders under a given path and documents correspond to all those files which have a timestamp as their prefix in the format yyyy_mm_dd_hh_mm_ss_mmm_*. All the derived classes must override the function data_loader(short_path,file_long_name) which determines how the data are loaded into the document of the stream. The files of the described format must never be deleted. The call update(up_to_timestamp) must not be called unless it is guaranteed that later no files with earlier timestamps are added.

data_loader(*short_path*, *file_info*)

file_filter(*sorted_file_names*)

get_results(*stream*, *time_interval*)
Must be overridden by deriving classes. 1. Calculates/receives the documents in the stream for the time interval given 2. Returns success or failure and the results (for some channels the values of kwargs can override the return process, e.g. introduce callbacks)

path = ''

update_streams(*up_to_timestamp*)
Deriving classes must override this function

static walk(*directory*, *level=1*)

class hyperstream.channels.file_channel.**FileDateTimeVersion**(*filename*, *split_char='_'*)
Bases: hyperstream.utils.containers.Printable

Simple class to hold file details along with the timestamp and version number from the filename. Uses semantic version.

is_python

1.1.1.7 hyperstream.channels.memory_channel module

class hyperstream.channels.memory_channel.**MemoryChannel**(*channel_id*)
Bases: *hyperstream.channels.base_channel.BaseChannel*

Channel whose data lives in memory

check_calculation_times()

create_stream(*stream_id*, *sandbox=None*)
Must be overridden by deriving classes, must create the stream according to the tool and return its unique identifier stream_id

get_results(*stream*, *time_interval*)
Calculates/receives the documents in the stream interval determined by the stream :param stream: The stream reference :param time_interval: The time interval :return: The sorted data items

get_stream_writer(*stream*)
Must be overridden by deriving classes, must return a function(document_collection) which writes all the given documents of the form (timestamp,data) from document_collection to the stream Example:

```
.. code-block:: python
```

if stream_id==1:

def f(document_collection):

for (timestamp,data) in document_collection: database[timestamp] = data

return(f)

else: raise Exception('No stream with id '+str(stream_id))

non_empty_streams

purge_all(*remove_definitions=False*)
Clears all streams in the channel - use with caution!

Returns None

purge_stream(*stream_id*, *remove_definition=False*, *sandbox=None*)
Clears all the data in a given stream and the calculated intervals

Parameters

- **stream_id** – The stream id
- **remove_definition** – Whether to remove the stream definition as well
- **sandbox** – The sandbox id

Returns None

update_streams(*up_to_timestamp*)
Deriving classes must override this function

class hyperstream.channels.memory_channel.**ReadOnlyMemoryChannel**(*channel_id*, *up_to_timestamp=datetime.datetime(1, 1, 1, 0, 0, tzinfo=<UTC>)*)

Bases: *hyperstream.channels.base_channel.BaseChannel*

An abstract channel with a read-only set of memory-based streams. By default it is constructed empty with the last update at MIN_DATE. New streams and documents within streams are created with the update(up_to_timestamp) method, which ensures that the channel is up to date until up_to_timestamp. No documents nor streams are ever deleted. Any deriving class must override update_streams(up_to_timestamp) which must update self.streams to be calculated until up_to_timestamp exactly. The data structure self.streams is a dict of streams indexed by stream_id, each stream is a list of tuples (timestamp,data), in no specific order. Names and identifiers are the same in this channel.

create_stream(*stream_id*, *sandbox=None*)
Must be overridden by deriving classes, must create the stream according to the tool and return its unique identifier stream_id

get_results(*stream*, *time_interval*)
Must be overridden by deriving classes. 1. Calculates/receives the documents in the stream for the time interval given 2. Returns success or failure and the results (for some channels the values of kwargs can override the return process, e.g. introduce callbacks)

get_stream_writer(*stream*)
Must be overridden by deriving classes, must return a function(document_collection) which writes all the given documents of the form (timestamp,data) from document_collection to the stream Example:

```
.. code-block:: python
```

if stream_id==1:

def f(document_collection):

for (timestamp,data) in document_collection: database[timestamp] = data

return(f)

else: raise Exception('No stream with id '+str(stream_id))

update_state(*up_to_timestamp*)
Call this function to ensure that the channel is up to date at the time of timestamp. I.e., all the streams that have been created before or at that timestamp are calculated exactly until up_to_timestamp.

update_streams(*up_to_timestamp*)
Deriving classes must override this function

1.1.1.8 hyperstream.channels.module_channel module

class hyperstream.channels.module_channel.**ModuleChannel**(*channel_id*, *path*, *up_to_timestamp=datetime.datetime(1, 1, 1, 0, 0, tzinfo=<UTC>)*)
Bases: *hyperstream.channels.file_channel.FileChannel*

A channel of module streams, the documents in the streams contain functions that can be called to import the respective module

data_loader(*short_path*, *tool_info*)

file_filter(*sorted_file_names*)

update_state(*up_to_timestamp*)
Call this function to ensure that the channel is up to date at the time of timestamp. I.e., all the streams that have been created before or at that timestamp are calculated exactly until up_to_timestamp.

versions = None

1.1.1.9 hyperstream.channels.tool_channel module

class hyperstream.channels.tool_channel.**ToolChannel**(*channel_id*, *path*, *up_to_timestamp=datetime.datetime(1, 1, 1, 0, 0, tzinfo=<UTC>)*)
Bases: *hyperstream.channels.module_channel.ModuleChannel*

Special case of the file/module channel to load the tools to execute other streams

get_results(*stream*, *time_interval*)
Must be overridden by deriving classes. 1. Calculates/receives the documents in the stream for the time interval given 2. Returns success or failure and the results (for some channels the values of kwargs can override the return process, e.g. introduce callbacks)

1.1.1.10 Module contents

1.1.2 hyperstream.itertools2 package

1.1.2.1 Submodules

1.1.2.2 hyperstream.itertools2.itertools2 module

hyperstream.itertools2.itertools2.**any_set**(*data*)

hyperstream.itertools2.itertools2.**count**(*data*)

hyperstream.itertools2.itertools2.**online_average**(*data*, *n=0*, *mean=0.0*)

hyperstream.itertools2.itertools2.**online_product**(*data*, *total=1.0*)

hyperstream.itertools2.itertools2.**online_sum**(*data*, *total=0.0*)

hyperstream.itertools2.itertools2.**online_variance**(*data*, *n=0*, *mean=0.0*, *m2=0.0*)

1.1.2.3 Module contents

1.1.3 hyperstream.models package

1.1.3.1 Submodules

1.1.3.2 hyperstream.models.factor module

class hyperstream.models.factor.**FactorDefinitionModel**(**args*, ***kwargs*)
Bases: mongoengine.document.EmbeddedDocument

alignment_node
A unicode string field.

factor_type
A unicode string field.

output_plate
An embedded document field - with a declared document_type. Only valid values are subclasses of EmbeddedDocument.

sinks
A list field that wraps a standard field, allowing multiple instances of the field to be used as a list in the database.

If using with ReferenceFields see: one-to-many-with-listfields

Note: Required means it cannot be empty - as the default for ListFields is []

sources
A list field that wraps a standard field, allowing multiple instances of the field to be used as a list in the database.

If using with ReferenceFields see: one-to-many-with-listfields

Note: Required means it cannot be empty - as the default for ListFields is []

splitting_node
A unicode string field.

tool
An embedded document field - with a declared document_type. Only valid values are subclasses of EmbeddedDocument.

class hyperstream.models.factor.**OutputPlateDefinitionModel**(**args*, ***kwargs*)
Bases: mongoengine.document.EmbeddedDocument

description
A unicode string field.

meta_data_id
A unicode string field.

plate_id
A unicode string field.

use_provided_values
Boolean field type.

New in version 0.1.2.

1.1.3.3 hyperstream.models.meta_data module

class hyperstream.models.meta_data.**MetaDataModel**(**args*, ***values*)
Bases: mongoengine.document.Document

exception DoesNotExist
Bases: mongoengine.errors.DoesNotExist

exception MultipleObjectsReturned
Bases: mongoengine.errors.MultipleObjectsReturned

data
A unicode string field.

id
A field wrapper around MongoDB's ObjectIds.

identifier

objects
The default QuerySet Manager.

Custom QuerySet Manager functions can extend this class and users can add extra queryset functionality. Any custom manager methods must accept a Document class as its first argument, and a QuerySet as its second argument.

The method function should return a QuerySet , probably the same one that was passed in, but modified in some way.

parent
A unicode string field.

tag
A unicode string field.

to_dict()

1.1.3.4 hyperstream.models.node module

class hyperstream.models.node.**NodeDefinitionModel**(**args*, ***kwargs*)
Bases: mongoengine.document.EmbeddedDocument

channel_id
A unicode string field.

plate_ids
A list field that wraps a standard field, allowing multiple instances of the field to be used as a list in the database.

If using with ReferenceFields see: one-to-many-with-listfields

Note: Required means it cannot be empty - as the default for ListFields is []

stream_name
A unicode string field.

1.1.3.5 hyperstream.models.plate module

class hyperstream.models.plate.**PlateDefinitionModel**(**args*, ***values*)
Bases: mongoengine.document.Document

exception DoesNotExist
Bases: mongoengine.errors.DoesNotExist

exception MultipleObjectsReturned
Bases: mongoengine.errors.MultipleObjectsReturned

complement
Boolean field type.

New in version 0.1.2.

description
A unicode string field.

id
A field wrapper around MongoDB's ObjectIds.

meta_data_id
A unicode string field.

objects
The default QuerySet Manager.

Custom QuerySet Manager functions can extend this class and users can add extra queryset functionality. Any custom manager methods must accept a Document class as its first argument, and a QuerySet as its second argument.

The method function should return a QuerySet , probably the same one that was passed in, but modified in some way.

parent_plate
A unicode string field.

plate_id
A unicode string field.

values
A list field that wraps a standard field, allowing multiple instances of the field to be used as a list in the database.

If using with ReferenceFields see: one-to-many-with-listfields

Note: Required means it cannot be empty - as the default for ListFields is []

class hyperstream.models.plate.**PlateModel**(**args*, ***kwargs*)
Bases: mongoengine.document.EmbeddedDocument

complement
Boolean field type.

New in version 0.1.2.

meta_data_id
A unicode string field.

values
A list field that wraps a standard field, allowing multiple instances of the field to be used as a list in the database.

If using with ReferenceFields see: one-to-many-with-listfields

Note: Required means it cannot be empty - as the default for ListFields is []

1.1.3.6 hyperstream.models.stream module

class hyperstream.models.stream.**StreamDefinitionModel**(**args*, ***values*)
Bases: mongoengine.document.Document

exception DoesNotExist
Bases: mongoengine.errors.DoesNotExist

exception MultipleObjectsReturned
Bases: mongoengine.errors.MultipleObjectsReturned

calculated_intervals
A ListField designed specially to hold a list of embedded documents to provide additional query helpers.

Note: The only valid list values are subclasses of EmbeddedDocument.

New in version 0.9.

channel_id
A unicode string field.

get_calculated_intervals()

id
A field wrapper around MongoDB's ObjectIds.

last_accessed
Datetime field.

Uses the python-dateutil library if available alternatively use time.strptime to parse the dates. Note: python-dateutil's parser is fully featured and when installed you can utilise it to convert varying types of date formats into valid python datetime objects.

Note: Microseconds are rounded to the nearest millisecond. Pre UTC microsecond support is effectively broken. Use ComplexDateTimeField if you need accurate microsecond support.

last_updated
Datetime field.

Uses the python-dateutil library if available alternatively use time.strptime to parse the dates. Note: python-dateutil's parser is fully featured and when installed you can utilise it to convert varying types of date formats into valid python datetime objects.

Note: Microseconds are rounded to the nearest millisecond. Pre UTC microsecond support is effectively broken. Use ComplexDateTimeField if you need accurate microsecond support.

objects
The default QuerySet Manager.

Custom QuerySet Manager functions can extend this class and users can add extra queryset functionality. Any custom manager methods must accept a `Document` class as its first argument, and a `QuerySet` as its second argument.

The method function should return a `QuerySet` , probably the same one that was passed in, but modified in some way.

sandbox
A unicode string field.

set_calculated_intervals(*intervals*)

stream_id
An embedded document field - with a declared document_type. Only valid values are subclasses of `EmbeddedDocument`.

stream_type
A unicode string field.

class hyperstream.models.stream.**StreamIdField**(**args*, ***kwargs*)
Bases: `mongoengine.document.EmbeddedDocument`

meta_data
A list field that wraps a standard field, allowing multiple instances of the field to be used as a list in the database.

If using with ReferenceFields see: one-to-many-with-listfields

Note: Required means it cannot be empty - as the default for ListFields is []

name
A unicode string field.

class hyperstream.models.stream.**StreamInstanceModel**(**args*, ***values*)
Bases: `mongoengine.document.Document`

exception DoesNotExist
Bases: `mongoengine.errors.DoesNotExist`

exception MultipleObjectsReturned
Bases: `mongoengine.errors.MultipleObjectsReturned`

datetime
Datetime field.

Uses the python-dateutil library if available alternatively use time.strptime to parse the dates. Note: python-dateutil's parser is fully featured and when installed you can utilise it to convert varying types of date formats into valid python datetime objects.

Note: Microseconds are rounded to the nearest millisecond. Pre UTC microsecond support is effectively broken. Use `ComplexDateTimeField` if you need accurate microsecond support.

id
A field wrapper around MongoDB's ObjectIds.

objects
The default QuerySet Manager.

Custom QuerySet Manager functions can extend this class and users can add extra queryset functionality. Any custom manager methods must accept a `Document` class as its first argument, and a `QuerySet` as its second argument.

The method function should return a `QuerySet` , probably the same one that was passed in, but modified in some way.

stream_id
An embedded document field - with a declared document_type. Only valid values are subclasses of `EmbeddedDocument`.

stream_type
A unicode string field.

value
A truly dynamic field type capable of handling different and varying types of data.

Used by `DynamicDocument` to handle dynamic data

1.1.3.7 hyperstream.models.time_interval module

class hyperstream.models.time_interval.**TimeIntervalModel**(**args*, ***kwargs*)
Bases: `mongoengine.document.EmbeddedDocument`

end
Datetime field.

Uses the python-dateutil library if available alternatively use time.strptime to parse the dates. Note: python-dateutil's parser is fully featured and when installed you can utilise it to convert varying types of date formats into valid python datetime objects.

Note: Microseconds are rounded to the nearest millisecond. Pre UTC microsecond support is effectively broken. Use `ComplexDateTimeField` if you need accurate microsecond support.

start
Datetime field.

Uses the python-dateutil library if available alternatively use time.strptime to parse the dates. Note: python-dateutil's parser is fully featured and when installed you can utilise it to convert varying types of date formats into valid python datetime objects.

Note: Microseconds are rounded to the nearest millisecond. Pre UTC microsecond support is effectively broken. Use `ComplexDateTimeField` if you need accurate microsecond support.

1.1.3.8 hyperstream.models.tool module

class hyperstream.models.tool.**ToolModel**(**args*, ***kwargs*)
Bases: `mongoengine.document.EmbeddedDocument`

name
A unicode string field.

parameters
A `ListField` designed specially to hold a list of embedded documents to provide additional query helpers.

Note: The only valid list values are subclasses of `EmbeddedDocument`.

New in version 0.9.

version
A unicode string field.

class hyperstream.models.tool.**ToolParameterModel**(**args*, ***kwargs*)
Bases: mongoengine.document.EmbeddedDocument

is_function
Boolean field type.

New in version 0.1.2.

is_set
Boolean field type.

New in version 0.1.2.

key
A unicode string field.

value
A truly dynamic field type capable of handling different and varying types of data.

Used by DynamicDocument to handle dynamic data

1.1.3.9 hyperstream.models.workflow module

class hyperstream.models.workflow.**WorkflowDefinitionModel**(**args*, ***values*)
Bases: mongoengine.document.Document

exception DoesNotExist
Bases: mongoengine.errors.DoesNotExist

exception MultipleObjectsReturned
Bases: mongoengine.errors.MultipleObjectsReturned

description
A unicode string field.

factors
A ListField designed specially to hold a list of embedded documents to provide additional query helpers.

Note: The only valid list values are subclasses of EmbeddedDocument.

New in version 0.9.

id
A field wrapper around MongoDB's ObjectIds.

monitor
Boolean field type.

New in version 0.1.2.

name
A unicode string field.

nodes
A `ListField` designed specially to hold a list of embedded documents to provide additional query helpers.

Note: The only valid list values are subclasses of `EmbeddedDocument`.

New in version 0.9.

objects
The default QuerySet Manager.

Custom QuerySet Manager functions can extend this class and users can add extra queryset functionality. Any custom manager methods must accept a `Document` class as its first argument, and a `QuerySet` as its second argument.

The method function should return a `QuerySet` , probably the same one that was passed in, but modified in some way.

online
Boolean field type.

New in version 0.1.2.

owner
A unicode string field.

workflow_id
A unicode string field.

class hyperstream.models.workflow.**WorkflowStatusModel**(**args*, ***values*)
Bases: `mongoengine.document.Document`

exception DoesNotExist
Bases: `mongoengine.errors.DoesNotExist`

exception MultipleObjectsReturned
Bases: `mongoengine.errors.MultipleObjectsReturned`

id
A field wrapper around MongoDB's ObjectIds.

last_accessed
Datetime field.

Uses the python-dateutil library if available alternatively use time.strptime to parse the dates. Note: python-dateutil's parser is fully featured and when installed you can utilise it to convert varying types of date formats into valid python datetime objects.

Note: Microseconds are rounded to the nearest millisecond. Pre UTC microsecond support is effectively broken. Use `ComplexDateTimeField` if you need accurate microsecond support.

last_updated
Datetime field.

Uses the python-dateutil library if available alternatively use time.strptime to parse the dates. Note: python-dateutil's parser is fully featured and when installed you can utilise it to convert varying types of date formats into valid python datetime objects.

Note: Microseconds are rounded to the nearest millisecond. Pre UTC microsecond support is effectively broken. Use `ComplexDateTimeField` if you need accurate microsecond support.

objects
The default QuerySet Manager.

Custom QuerySet Manager functions can extend this class and users can add extra queryset functionality. Any custom manager methods must accept a `Document` class as its first argument, and a `QuerySet` as its second argument.

The method function should return a `QuerySet` , probably the same one that was passed in, but modified in some way.

requested_intervals
A `ListField` designed specially to hold a list of embedded documents to provide additional query helpers.

Note: The only valid list values are subclasses of `EmbeddedDocument`.

New in version 0.9.

workflow_id
A unicode string field.

1.1.3.10 Module contents

1.1.4 hyperstream.stream package

1.1.4.1 Submodules

1.1.4.2 hyperstream.stream.stream module

class hyperstream.stream.stream.**AssetStream**(*channel*, *stream_id*, *calculated_intervals*, *last_accessed*, *last_updated*, *sandbox*, *mongo_model=None*)
Bases: *hyperstream.stream.stream.DatabaseStream*

Simple subclass that overrides the calculated intervals property

calculated_intervals

class hyperstream.stream.stream.**DatabaseStream**(*channel*, *stream_id*, *calculated_intervals*, *last_accessed*, *last_updated*, *sandbox*, *mongo_model=None*)
Bases: *hyperstream.stream.stream.Stream*

Simple subclass that overrides the calculated intervals property

calculated_intervals
Gets the calculated intervals from the database

Returns The calculated intervals

last_accessed
Gets the last accessed time from the database

Returns The last accessed time

last_updated
Gets the last updated time from the database

Returns The last updated time

load()
Load the stream definition from the database

Returns None

save()
Saves the stream definition to the database. This assumes that the definition doesn't already exist, and will raise an exception if it does.

Returns None

class hyperstream.stream.stream.**Stream**(*channel*, *stream_id*, *calculated_intervals*, *sandbox*)
Bases: hyperstream.utils.containers.Hashable

Stream reference class

calculated_intervals
Get the calculated intervals This will be read from the stream_status collection if it's in the database channel

Returns The calculated intervals

parent_node

purge()
Purge the stream. This removes all data and clears the calculated intervals

Returns None

set_tool_reference(*tool_reference*)
Set the back reference to the tool that populates this stream. This is needed to traverse the graph outside of workflows

Parameters **tool_reference** – The toool

Returns None

window(*time_interval=None*, *force_calculation=False*)
Gets a view on this stream for the time interval given

Parameters

- **time_interval** (*None | Iterable | TimeInterval*) – either a TimeInterval object or (start, end) tuple of type str or datetime
- **force_calculation** (*bool*) – Whether we should force calculation for this stream view if data does not exist

Returns a stream view object

writer

1.1.4.3 hyperstream.stream.stream_collections module

class hyperstream.stream.stream_collections.**StreamDict**(**args*, ***kwargs*)
Bases: hyperstream.utils.containers.TypedBiDict

Custom bi-directional dictionary where keys are StreamID objects and values are Stream objects. Raises ValueDuplicationError if the same Stream is added again

class hyperstream.stream.stream_collections.**StreamInstanceCollection**
Bases: hyperstream.utils.containers.FrozenKeyDict

A custom frozen dictionary for stream instances. Will raise an exception if a repeated instance is added

append(*instance*)

extend(*instances*)

1.1.4.4 hyperstream.stream.stream_id module

class hyperstream.stream.stream_id.**StreamId**(*name*, *meta_data=None*)
Bases: hyperstream.utils.containers.Hashable

Helper class for stream identifiers. A stream identifier contains the stream name and any meta-data

as_dict()

as_raw()
Return a representation of this object that can be used with mongoengine Document.objects(__raw__=x) Example:

```
>>> stream_id = StreamId(name='test', meta_data=((u'house', u'1'), (u'resident
↪', u'1')))
>>> stream_id.as_raw()
{'stream_id.meta_data': [(u'house', u'1'), (u'resident', u'1')], 'stream_id.
↪name': 'test'}
```

Returns The raw representation of this object.

to_json()

hyperstream.stream.stream_id.**get_stream_id**(*item*)

1.1.4.5 hyperstream.stream.stream_instance module

class hyperstream.stream.stream_instance.**StreamInstance**
Bases: *hyperstream.stream.stream_instance.StreamInstance*

Simple helper class for storing data instances that's a bit neater than simple tuples

as_dict(*flat=True*)

as_list(*flat=True*)

class hyperstream.stream.stream_instance.**StreamMetaInstance**
Bases: *hyperstream.stream.stream_instance.StreamMetaInstance*

StreamInstance that also contains meta data

1.1.4.6 hyperstream.stream.stream_view module

class hyperstream.stream.stream_view.**StreamView**(*stream*, *time_interval*, *force_calculation=False*)
Bases: hyperstream.utils.containers.Printable

Simple helper class for storing streams with a time interval (i.e. a "view" on a stream) :param stream: The stream upon which this is a view :param time_interval: The time interval over which this view is defined :param

force_calculation: Whether we should force calculation for this stream view if data does not exist :type stream: Stream :type time_interval: TimeInterval

component(*key*)

component_filter(*key*, *values*)

delete_nones()

dict_items(*flat=True*)

dict_iteritems(*flat=True*)

first(*default=None*)

head(*n*)

islice(*start*, *stop=None*, *step=1*)

items()
Return all results as a list :return: The results :rtype: list[StreamInstance]

iteritems()

itertimestamps()

itervalues()

last(*default=None*)

tail(*n*)

timestamps()

values()

1.1.4.7 Module contents

1.1.5 hyperstream.tool package

1.1.5.1 Submodules

1.1.5.2 hyperstream.tool.aggregate_tool module

class hyperstream.tool.aggregate_tool.**AggregateTool**(*aggregation_meta_data*, ***kwargs*)
Bases: *hyperstream.tool.tool.Tool*

This type of tool aggregates over a given plate. For example, if the input is all the streams in a node on plate A.B, and the aggregation is over plate B, the results will live on plate A alone. This can also be thought of as marginalising one dimension of a tensor over the plates

1.1.5.3 hyperstream.tool.base_tool module

class hyperstream.tool.base_tool.**BaseTool**(***kwargs*)
Bases: hyperstream.utils.containers.Printable, hyperstream.utils.containers.Hashable

Base class for all tools

get_model()
Gets the mongoengine model for this tool, which serializes parameters that are functions

Returns The mongoengine model. TODO: Note that the tool version is currently incorrect (0.0.0)

message(*interval*)
Get the execution message

Parameters **interval** – The time interval

Returns The execution message

name
Get the name of the tool, converted to snake format (e.g. "splitter_from_stream")

Returns The name

parameters
Get the tool parameters

Returns The tool parameters along with additional information (whether they are functions or sets)

parameters_dict
Get the tool parameters as a simple dictionary

Returns The tool parameters

static parameters_from_dicts(*parameters*)
Get the tool parameters model from dictionaries

Parameters **parameters** – The parameters as dictionaries

Returns The tool parameters model

static parameters_from_model(*parameters_model*)
Get the tool parameters model from dictionaries

Parameters **parameters_model** – The parameters as a mongoengine model

Returns The tool parameters as a dictionary

static write_to_history(**kwargs*)
Write to the history of executions of this tool

Parameters **kwargs** – keyword arguments describing the executions

Returns None

1.1.5.4 hyperstream.tool.multi_output_tool module

class hyperstream.tool.multi_output_tool.**MultiOutputTool**(**kwargs*)
Bases: *hyperstream.tool.base_tool.BaseTool*

Special type of tool that outputs multiple streams on a new plate rather than a single stream. There are in this case multiple sinks rather than a single sink, and a single source rather than multiple sources. Note that no alignment stream is required here. Also note that we don't subclass Tool due to different calling signatures

execute(*source*, *splitting_stream*, *sinks*, *interval*, *meta_data_id*, *output_plate_values*)
Execute the tool over the given time interval.

Parameters

- **source** (Stream) – The source stream
- **splitting_stream** – The stream over which to split

- **sinks** (*list[Stream] | tuple[Stream]*) – The sink streams
- **interval** (TimeInterval) – The time interval
- **meta_data_id** (*str*) – The meta data id of the output plate
- **output_plate_values** (*list | tuple*) – The values of the plate where data is put onto

Returns None

1.1.5.5 hyperstream.tool.plate_creation_tool module

class hyperstream.tool.plate_creation_tool.**PlateCreationTool**(***kwargs*)
Bases: *hyperstream.tool.base_tool.BaseTool*

Special type of tool that creates a new plate. There is no sink in this case, as it does not yet exist. Note that no alignment stream is required here. Also note that we don't subclass Tool due to different calling signatures

execute(*source*, *interval*, *input_plate_value*)
Execute the tool over the given time interval.

Parameters

- **source** (Stream) – The source stream
- **interval** (TimeInterval) – The time interval
- **input_plate_value** (*tuple[tuple[str, str]] | None*) – The value of the plate where data comes from (can be None)

Returns None

1.1.5.6 hyperstream.tool.selector_tool module

class hyperstream.tool.selector_tool.**SelectorTool**(*selector_meta_data*, ***kwargs*)
Bases: *hyperstream.tool.base_tool.BaseTool*

This type of tool performs sub-selection of streams within a node. This can either be done using a selector in the parameters or using an input stream. The sink node plate should be a sub-plate of the source node. Examples are IndexOf and SubArray, either with fixed or variable parameters

execute(*sources*, *sinks*, *interval*)
Execute the tool over the given time interval.

Parameters

- **sources** (*list[Stream] | tuple[Stream]*) – The source streams
- **sinks** (*list[Stream] | tuple[Stream]*) – The sink streams
- **interval** (TimeInterval) – The time interval

Returns None

1.1.5.7 hyperstream.tool.tool module

class hyperstream.tool.tool.**Tool**(***kwargs*)
Bases: *hyperstream.tool.base_tool.BaseTool*

Base class for tools. Tools are the unit of computation, operating on input streams to produce an output stream

execute(*sources*, *sink*, *interval*, *alignment_stream=None*)
Execute the tool over the given time interval. If an alignment stream is given, the output instances will be aligned to this stream

Parameters

- **sources** (*list[Stream] | tuple[Stream] | None*) – The source streams (possibly None)
- **sink** (Stream) – The sink stream
- **alignment_stream** (*Stream | None*) – The alignment stream
- **interval** (TimeInterval) – The time interval

Returns None

1.1.5.8 Module contents

Tool package. Defines Tool, MultiOutputTool and SelectorTool base classes.

1.1.6 hyperstream.utils package

1.1.6.1 Submodules

1.1.6.2 hyperstream.utils.decorators module

hyperstream.utils.decorators.**check_input_stream_count**(*expected_number_of_streams*)
Decorator for Tool._execute that checks the number of input streams

Parameters **expected_number_of_streams** – The expected number of streams

Returns the decorator

hyperstream.utils.decorators.**check_output_format**(*expected_formats*)
Decorator for stream outputs that checks the format of the outputs after modifiers have been applied :param expected_formats: The expected output formats :type expected_formats: tuple, set :return: the decorator

hyperstream.utils.decorators.**check_tool_defined**(*func*)
Decorator to check whether a tool stream has been defined before execution :return: the decorator

hyperstream.utils.decorators.**timeit**(*f*)

1.1.6.3 hyperstream.utils.errors module

exception hyperstream.utils.errors.**ChannelAlreadyExistsError**
Bases: exceptions.Exception

exception hyperstream.utils.errors.**ChannelNotFoundError**
Bases: exceptions.Exception

exception hyperstream.utils.errors.**ConfigurationError**
Bases: exceptions.Exception

exception hyperstream.utils.errors.**FactorAlreadyExistsError**
Bases: exceptions.Exception

message = 'Cannot have duplicate factors - a new factor object should be created'

exception hyperstream.utils.errors.**FactorDefinitionError**
Bases: exceptions.Exception

exception hyperstream.utils.errors.**IncompatiblePlatesError**
Bases: exceptions.Exception

exception hyperstream.utils.errors.**IncompatibleToolError**
Bases: exceptions.Exception

exception hyperstream.utils.errors.**LinkageError**
Bases: exceptions.Exception

exception hyperstream.utils.errors.**MultipleStreamsFoundError**
Bases: exceptions.Exception

exception hyperstream.utils.errors.**NodeAlreadyExistsError**
Bases: exceptions.Exception

message = 'Cannot have duplicate nodes'

exception hyperstream.utils.errors.**NodeDefinitionError**
Bases: exceptions.Exception

exception hyperstream.utils.errors.**PlateDefinitionError**
Bases: exceptions.Exception

message = 'Empty values in plate definition and complement=False'

exception hyperstream.utils.errors.**PlateEmptyError**(*plate_id*)
Bases: exceptions.Exception

message = 'Plate values for {} empty'

exception hyperstream.utils.errors.**PlateNotFoundError**
Bases: exceptions.Exception

exception hyperstream.utils.errors.**StreamAlreadyExistsError**
Bases: exceptions.Exception

exception hyperstream.utils.errors.**StreamDataNotAvailableError**
Bases: exceptions.Exception

exception hyperstream.utils.errors.**StreamNotAvailableError**(*up_to_timestamp*)
Bases: exceptions.Exception

message = 'The stream is not available after {} and cannot be calculated'

exception hyperstream.utils.errors.**StreamNotFoundError**
Bases: exceptions.Exception

exception hyperstream.utils.errors.**ToolExecutionError**(*required_intervals*)
Bases: exceptions.Exception

message = 'Tool execution did not cover the time interval {}.'

exception hyperstream.utils.errors.**ToolInitialisationError**
Bases: exceptions.Exception

exception hyperstream.utils.errors.**ToolNotFoundError**
Bases: exceptions.Exception

hyperstream.utils.errors.**handle_exception**(*exc_type*, *exc_value*, *exc_traceback*)

1.1.6.4 hyperstream.utils.serialization module

hyperstream.utils.serialization.**func_dump**(*func*)
Serialize user defined function. :param func: The function :return: Tuple of code, defaults and closure

hyperstream.utils.serialization.**func_load**(*code*, *defaults=None*, *closure=None*, *globs=None*)
Reload a function :param code: The code object :param defaults: Default values :param closure: The closure :param globs: globals :return:

hyperstream.utils.serialization.**func_reconstruct_closure**(*values*)
Deserialization helper that reconstructs a closure :param values: The closure values :return: The closure

1.1.6.5 hyperstream.utils.time_utils module

hyperstream.utils.time_utils.**construct_experiment_id**(*time_interval*)
Construct an experiment id from a time interval :return: The experiment id :type time_interval: TimeInterval :rtype: str

hyperstream.utils.time_utils.**datetime2unix**(*dt*)

hyperstream.utils.time_utils.**duration2str**(*x*)

hyperstream.utils.time_utils.**get_timedelta**(*value*)

hyperstream.utils.time_utils.**is_naive**(*dt*)

hyperstream.utils.time_utils.**json_serial**(*obj*)
JSON serializer for objects not serializable by default json code

hyperstream.utils.time_utils.**reconstruct_interval**(*experiment_id*)
Reverse the construct_experiment_id operation :param experiment_id: The experiment id :return: time interval

hyperstream.utils.time_utils.**remove_microseconds**(*ts*)

hyperstream.utils.time_utils.**unix2datetime**(*u*)

hyperstream.utils.time_utils.**utcnow**()
Gets the current datetime in UTC format with millisecond precision :return:

1.1.6.6 hyperstream.utils.utils module

1.1.6.7 Module contents

1.1.7 hyperstream.workflow package

1.1.7.1 Submodules

1.1.7.2 hyperstream.workflow.factor module

1.1.7.3 hyperstream.workflow.meta_data_manager module

1.1.7.4 hyperstream.workflow.node module

1.1.7.5 hyperstream.workflow.plate module

1.1.7.6 hyperstream.workflow.plate_manager module

1.1.7.7 hyperstream.workflow.workflow module

Workflow and WorkflowMonitor definitions.

class hyperstream.workflow.workflow.**Workflow**(*workflow_id*, *name*, *description*, *owner*, *online=False*, *monitor=False*)

Bases: hyperstream.utils.containers.Printable

Workflow. This defines the graph of operations through "nodes" and "factors".

static check_multi_output_plate_compatibility(*source_plates*, *sink_plate*)

Check multi-output plate compatibility. This ensures that the source plates and sink plates match for a multi- output plate

Parameters

- **source_plates** – The source plates
- **sink_plate** – The sink plate

Returns True if the plates are compatible

static check_plate_compatibility(*tool*, *source_plate*, *sink_plate*)

Checks whether the source and sink plate are compatible given the tool

Parameters

- **tool** (Tool) – The tool
- **source_plate** (*Plate*) – The source plate
- **sink_plate** (*Plate*) – The sink plate

Returns Either an error, or None

Return type None | str

create_factor(*tool*, *sources*, *sink*, *alignment_node=None*)

Creates a factor. Instantiates a single tool for all of the plates, and connects the source and sink nodes with that tool.

Note that the tool parameters these are currently fixed over a plate. For parameters that vary over a plate, an extra input stream should be used

Parameters

- **alignment_node** (*Node | None*) –
- **tool** (*Tool | dict*) – The tool to use. This is either an instantiated Tool object or a dict with "name" and "parameters"
- **sources** (*list[Node] | tuple[Node] | None*) – The source nodes
- **sink** (*Node*) – The sink node

Returns The factor object

Return type Factor

create_factor_general(**args*, ***kwargs*)

General signature for factor creation that tries each of the factor creation types using duck typing

Parameters

- **args** – The positional arguments
- **kwargs** – The named arguments

Returns The created factor

create_multi_output_factor(*tool*, *source*, *splitting_node*, *sink*)

Creates a multi-output factor. This takes a single node, applies a MultiOutputTool to create multiple nodes on a new plate Instantiates a single tool for all of the input plate values, and connects the source and sink nodes with that tool.

Note that the tool parameters these are currently fixed over a plate. For parameters that vary over a plate, an extra input stream should be used

Parameters

- **tool** (*MultiOutputTool | dict*) – The tool to use. This is either an instantiated Tool object or a dict with "name" and "parameters"
- **source** (*Node | None*) – The source node
- **splitting_node** – The node over which to split
- **sink** (*Node*) – The sink node

Returns The factor object

Return type Factor

create_node(*stream_name*, *channel*, *plates*)

Create a node in the graph. Note: assumes that the streams already exist

Parameters

- **stream_name** – The name of the stream
- **channel** – The channel where this stream lives
- **plates** – The plates. The stream meta-data will be auto-generated from these

Returns The streams associated with this node

create_node_creation_factor(*tool*, *source*, *output_plate*, *plate_manager*)

Creates a factor that itself creates an output node, and ensures that the plate for the output node exists along with all relevant meta-data

Parameters

- **tool** – The tool
- **source** – The source node
- **output_plate** (*dict*) – The details of the plate that will be created (dict)
- **plate_manager** (*PlateManager*) – The hyperstream plate manager

Returns The created factor

execute(*time_interval*)
Here we execute the factors over the streams in the workflow Execute the factors in reverse order. We can't just execute the last factor because there may be multiple "leaf" factors that aren't triggered by upstream computations.

Parameters **time_interval** – The time interval to execute this workflow over

static factorgraph_viz(*d*)
Map the dictionary into factorgraph-viz format. See https://github.com/mbforbes/factorgraph-viz

Parameters **d** – The dictionary

Returns The formatted dictionary

requested_intervals
Get the requested intervals (from the database)

Returns The requested intervals

to_dict(*tool_long_names=True*)
Get a representation of the workflow as a dictionary for display purposes

Parameters **tool_long_names** (*bool*) – Indicates whether to use long names, such as SplitterFromStream(element=None, use_mapping_keys_only=True) or short names, such as splitter_from_stream

Returns The dictionary of nodes, factors and plates

to_json(*formatter=None*, *tool_long_names=True*, ***kwargs*)
Get a JSON representation of the workflow

Parameters

- **tool_long_names** – Indicates whether to use long names, such as SplitterFromStream(element=None, use_mapping_keys_only=True) or short names, such as splitter_from_stream
- **formatter** – The formatting function
- **kwargs** – Keyword arguments for the json output

Returns A JSON string

class hyperstream.workflow.workflow.**WorkflowMonitor**(*workflow*)
Bases: object

Small helper class that provides logging output to monitor workflow progress

1.1.7.8 hyperstream.workflow.workflow_manager module

class hyperstream.workflow.workflow_manager.**WorkflowManager**(*channel_manager*, *plate_manager*)
Bases: hyperstream.utils.containers.Printable

Workflow manager. Responsible for reading and writing workflows to the database, and can execute all of the workflows

add_workflow(*workflow*, *commit=False*)
Add a new workflow and optionally commit it to the database :param workflow: The workflow :param commit: Whether to commit the workflow to the database :type workflow: Workflow :type commit: bool :return: None

commit_all()
Commit all workflows to the database :return: None

commit_workflow(*workflow_id*)
Commit the workflow to the database :param workflow_id: The workflow id :return: None

delete_workflow(*workflow_id*)
Delete a workflow from the database :param workflow_id: :return: None

execute_all()
Execute all workflows

load_workflow(*workflow_id*)
Load workflow from the database and store in memory :param workflow_id: The workflow id :return: The workflow

set_all_requested_intervals(*requested_intervals*)
Sets the requested intervals for all workflow :param requested_intervals: The requested intervals :return: None :type requested_intervals: TimeIntervals

set_requested_intervals(*workflow_id*, *requested_intervals*)
Sets the requested intervals for a given workflow :param workflow_id: The workflow id :param requested_intervals: The requested intervals :return: None :type requested_intervals: TimeIntervals

hyperstream.workflow.workflow_manager.**code_pickler**(*code*)

hyperstream.workflow.workflow_manager.**code_unpickler**(*data*)

1.1.7.9 Module contents

1.2 Submodules

1.3 hyperstream.channel_manager module

1.4 hyperstream.client module

The main hyperstream client connection that is used for storing runtime information. Note that this is also used by the default database channel, although other database channels (connecting to different database types) can also be used.

class hyperstream.client.**Client**(*server_config*, *auto_connect=True*)
Bases: hyperstream.utils.containers.Printable

The main mongo client

client = None

connect(*server_config*)
Connect using the configuration given

Parameters **server_config** – The server configuration

db = None

get_config_value(*key*, *default=None*)
Get a specific value from the configuration

Parameters

- **key** – The of the item
- **default** – A default value if not found

Returns The found value or the default

session = None

1.5 hyperstream.config module

HyperStream configuration module.

class hyperstream.config.**HyperStreamConfig**(*filename='hyperstream_config.json'*)
Bases: hyperstream.utils.containers.Printable

Wrapper around the hyperstream configuration file

class hyperstream.config.**OnlineEngineConfig**(*interval*, *sleep=5*, *iterations=100*, *alarm=None*)
Bases: hyperstream.utils.containers.Printable

1.6 hyperstream.hyperstream module

Main HyperStream class

class hyperstream.hyperstream.**HyperStream**(*loglevel=20*, *file_logger=True*, *console_logger=True*, *mqtt_logger=None*, *config_filename='hyperstream_config.json'*)
Bases: object

HyperStream class: can be instantiated simply with hyperstream = HyperStream() for default operation. Use in the following way to create a session (and store history of execution etc). >>> with Hyperstream(): >>> pass

Note that HyperStream uses the singleton pattern described here: https://stackoverflow.com/a/33201/1038264

For py2k/py3k compatability we use the six decorator add_metaclass: https://pythonhosted.org/six/#six.add_metaclass

add_workflow(*workflow*)
Add the workflow to the workflow manager

Parameters **workflow** (Workflow) – The workflow

Returns None

clear_sessions(*inactive_only=True*, *clear_history=False*)
Clears all stored sessions, optionally excluding active sessions

Parameters

- **inactive_only** – Whether to clear inactive sessions only
- **clear_history** – Whether to also clear session history

Returns None

create_workflow(***kwds*)
Create a new workflow. Simple wrapper for creating a workflow and adding it to the workflow manager.

Parameters

- **workflow_id** – The workflow id
- **name** – The workflow name
- **owner** – The owner/creator of the workflow
- **description** – A human readable description
- **online** – Whether this workflow should be executed by the online engine
- **monitor** – Whether the workflow computations should be monitored
- **safe** – If safe=True, will throw an error if the workflow already exists

Returns The workflow

current_session
Get the current session

Returns the current session

new_session()
Start a new session to record computation history

Returns the created session

populate_tools_and_factors()
Function to populate factory functions for the tools and factors for ease of access.

Returns None

sessions
Get the list of sessions

Returns the sessions

1.7 hyperstream.online_engine module

Online Engine module. This will be used in the online execution mode.

class hyperstream.online_engine.**OnlineEngine**(*hyperstream*)
Bases: object

OnlineEngine class.

execute(***kwargs*)
Execute the engine - currently simple executes all workflows.

1.8 hyperstream.plugin_manager module

Plugin manager module for additional user added channels and tools.

class hyperstream.plugin_manager.**Plugin**
Bases: hyperstream.plugin_manager.PluginBase, hyperstream.utils.containers.Printable

Plugin class - simple wrapper over namedtuple

load_channels()
Loads the channels and tools given the plugin path specified

Returns The loaded channels, including a tool channel, for the tools found.

1.9 hyperstream.time_interval module

Module for dealing with time intervals containing TimeInterval, TimeIntervals, and RelativeTimeInterval

class hyperstream.time_interval.**RelativeTimeInterval**(*start*, *end*)
Bases: *hyperstream.time_interval.TimeInterval*

Relative time interval object. Thin wrapper around a (start, end) tuple of timedelta objects that provides some validation

absolute(*dt*)

end

start

class hyperstream.time_interval.**TimeInterval**(*start*, *end*)
Bases: *hyperstream.time_interval.TimeInterval*

Time interval object. Thin wrapper around a (start, end) tuple of datetime objects that provides some validation

classmethod all_time()

end

humanized

classmethod now_minus(*weeks=0*, *days=0*, *hours=0*, *minutes=0*, *seconds=0*, *milliseconds=0*)

start

to_json()

to_tuple()

classmethod up_to_now()

width

class hyperstream.time_interval.**TimeIntervals**(*intervals=None*)
Bases: hyperstream.utils.containers.Printable

Container class for time intervals, that manages splitting and joining Example object: (t1,t2] U (t3,t4] U ...

compress()

end

humanized

is_empty

parse(*intervals*)

span

split(*points*)
Splits the list of time intervals in the specified points

The function assumes that the time intervals do not overlap and ignores points that are not inside of any interval.

Parameters **points**(*list of datetime*) –

start

to_json()

hyperstream.time_interval.**parse_time_tuple**(*start*, *end*)

Parse a time tuple. These can be: relative in seconds, e.g. (-4, 0) relative in timedelta, e.g. (timedelta(seconds=-4), timedelta(0)) absolute in date/datetime, e.g. (datetime(2016, 4, 28, 20, 0, 0, 0, UTC), datetime(2016, 4, 28, 21, 0, 0, 0, UTC)) absolute in iso strings, e.g. ("2016-04-28T20:00:00.000Z", "2016-04-28T20:01:00.000Z") Mixtures of relative and absolute are not allowed

Parameters

- **start**(*int | timedelta | datetime | str*) – Start time
- **end**(*int | timedelta | datetime | str*) – End time

Returns TimeInterval or RelativeTimeInterval object

hyperstream.time_interval.**profile**(*ob*)
Comment out this function to be able to use the line_profiler module. e.g. call: kernprof -l scripts/deploy_summariser.py –loglevel=10 python -m line_profiler deploy_summariser.py.lprof > deploy_summariser.py.summary :param ob: object :return: object

1.10 hyperstream.version module

1.11 Module contents

CHAPTER 2

tests package

2.1 Submodules

2.2 tests.helpers module

class tests.helpers.**MqttClient**
Bases: object

last_messages = {}

on_connect(*client*, *userdata*, *flags*, *rc*)

on_message(*client*, *userdata*, *msg*)

tests.helpers.**assert_all_close**(*a*, *b*, *tolerance*)

tests.helpers.**assert_dict_equal**(*a*, *b*)

tests.helpers.**create_plate**(*hs*, *plate_id*, *tag*, *parent_plate=None*)

tests.helpers.**delete_meta_data**(*hs*, *tag*, *values*, *parent='root'*)

tests.helpers.**delete_plate**(*hs*, *plate_id*)

tests.helpers.**get_meta_data**(*hs*, *tag*)

tests.helpers.**insert_meta_data**(*hs*, *tag*, *values*, *parent='root'*)

tests.helpers.**is_close**(*a*, *b*, *tolerance*)

tests.helpers.**mosquitto_is_running**()

tests.helpers.**resource_manager**(**args*, ***kwds*)

tests.helpers.**setup**()

tests.helpers.**teardown**()

2.3 tests.test_time_interval module

class tests.test_time_interval.**HyperStreamTimeIntervalTests**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_constructors()

test_relative_time_interval()

test_split()

test_time_interval()

2.4 tests.test_tool_channel module

2.5 Module contents

Running the tests:

Run the following command

```
>>> nosetests
```

Note that for the MQTT logging test to succeed, you will need to have an MQTT broker running (e.g. Mosquitto). For example:

` docker run -ti -p 1883:1883 -p 9001:9001 toke/mosquitto `

or on OSX you will need pidof and mosquitto:

` brew install pidof brew install mosquitto brew services start mosquitto `

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

h

t

Index

D

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W