

---

# HyperStream Documentation

*Release 0.3.7*

**SPHERE WP5**

**Aug 21, 2017**



<b>1</b>	<b>hyperstream package</b>	<b>3</b>
1.1	Subpackages	3
1.1.1	hyperstream.channels package	3
1.1.1.1	Submodules	3
1.1.1.2	hyperstream.channels.assets_channel module	3
1.1.1.3	hyperstream.channels.assets_channel2 module	4
1.1.1.4	hyperstream.channels.base_channel module	4
1.1.1.5	hyperstream.channels.database_channel module	5
1.1.1.6	hyperstream.channels.file_channel module	6
1.1.1.7	hyperstream.channels.memory_channel module	7
1.1.1.8	hyperstream.channels.module_channel module	8
1.1.1.9	hyperstream.channels.tool_channel module	8
1.1.1.10	Module contents	9
1.1.2	hyperstream.itertools2 package	9
1.1.2.1	Submodules	9
1.1.2.2	hyperstream.itertools2.itertools2 module	9
1.1.2.3	Module contents	9
1.1.3	hyperstream.models package	9
1.1.3.1	Submodules	9
1.1.3.2	hyperstream.models.factor module	9
1.1.3.3	hyperstream.models.meta_data module	10
1.1.3.4	hyperstream.models.node module	11
1.1.3.5	hyperstream.models.plate module	11
1.1.3.6	hyperstream.models.stream module	12
1.1.3.7	hyperstream.models.time_interval module	14
1.1.3.8	hyperstream.models.tool module	15
1.1.3.9	hyperstream.models.workflow module	15
1.1.3.10	Module contents	17
1.1.4	hyperstream.stream package	17
1.1.4.1	Submodules	17
1.1.4.2	hyperstream.stream.stream module	17
1.1.4.3	hyperstream.stream.stream_collections module	19
1.1.4.4	hyperstream.stream.stream_id module	19
1.1.4.5	hyperstream.stream.stream_instance module	19
1.1.4.6	hyperstream.stream.stream_view module	20
1.1.4.7	Module contents	20

1.1.5	hyperstream.tool package . . . . .	20
1.1.5.1	Submodules . . . . .	20
1.1.5.2	hyperstream.tool.aggregate_tool module . . . . .	20
1.1.5.3	hyperstream.tool.base_tool module . . . . .	21
1.1.5.4	hyperstream.tool.multi_output_tool module . . . . .	21
1.1.5.5	hyperstream.tool.plate_creation_tool module . . . . .	22
1.1.5.6	hyperstream.tool.selector_tool module . . . . .	22
1.1.5.7	hyperstream.tool.tool module . . . . .	23
1.1.5.8	Module contents . . . . .	23
1.1.6	hyperstream.utils package . . . . .	23
1.1.6.1	Submodules . . . . .	23
1.1.6.2	hyperstream.utils.decorators module . . . . .	23
1.1.6.3	hyperstream.utils.errors module . . . . .	24
1.1.6.4	hyperstream.utils.serialization module . . . . .	25
1.1.6.5	hyperstream.utils.time_utils module . . . . .	25
1.1.6.6	hyperstream.utils.utils module . . . . .	26
1.1.6.7	Module contents . . . . .	26
1.1.7	hyperstream.workflow package . . . . .	26
1.1.7.1	Submodules . . . . .	26
1.1.7.2	hyperstream.workflow.factor module . . . . .	26
1.1.7.3	hyperstream.workflow.meta_data_manager module . . . . .	26
1.1.7.4	hyperstream.workflow.node module . . . . .	26
1.1.7.5	hyperstream.workflow.plate module . . . . .	26
1.1.7.6	hyperstream.workflow.plate_manager module . . . . .	26
1.1.7.7	hyperstream.workflow.workflow module . . . . .	26
1.1.7.8	hyperstream.workflow.workflow_manager module . . . . .	28
1.1.7.9	Module contents . . . . .	29
1.2	Submodules . . . . .	29
1.3	hyperstream.channel_manager module . . . . .	29
1.4	hyperstream.client module . . . . .	29
1.5	hyperstream.config module . . . . .	30
1.6	hyperstream.hyperstream module . . . . .	30
1.7	hyperstream.online_engine module . . . . .	31
1.8	hyperstream.plugin_manager module . . . . .	31
1.9	hyperstream.time_interval module . . . . .	32
1.10	hyperstream.version module . . . . .	33
1.11	Module contents . . . . .	33
<b>2</b>	<b>tests package</b> . . . . .	<b>35</b>
2.1	Submodules . . . . .	35
2.2	tests.helpers module . . . . .	35
2.3	tests.test_time_interval module . . . . .	36
2.4	tests.test_tool_channel module . . . . .	36
2.5	Module contents . . . . .	36
<b>3</b>	<b>Indices and tables</b> . . . . .	<b>37</b>
	<b>Python Module Index</b> . . . . .	<b>39</b>

**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:



### Subpackages

#### hyperstream.channels package

##### Submodules

##### 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

## hyperstream.channels.assets\_channel2 module

## 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

## 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****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)
path = ''
update_streams (up_to_timestamp)
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

```

### 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)

```

```

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)

```

```
class hyperstream.channels.memory_channel.ReadOnlyMemoryChannel (channel_id,
                                                                up_to_timestamp=datetime.datetime(1,
                                                                1, 1, 0, 0, tz-
                                                                info=<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)

**get\_results** (stream, time\_interval)

**get\_stream\_writer** (stream)

**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

## 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)

**versions** = None

## 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)

## Module contents

### hyperstream.itertools2 package

#### Submodules

#### 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*)

## Module contents

### hyperstream.models package

#### Submodules

#### 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.

## hyperstream.models.meta\_data module

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

**exception DoesNotExist**

Bases: `mongoengine.errors.DoesNotExist`

**exception MetaDataModel.MultipleObjectsReturned**

Bases: `mongoengine.errors.MultipleObjectsReturned`

`MetaDataModel.data`

A unicode string field.

`MetaDataModel.id`

A field wrapper around MongoDB's ObjectIds.

`MetaDataModel.identifier`

`MetaDataModel.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.

`MetaDataModel.parent`

A unicode string field.

`MetaDataModel.tag`

A unicode string field.

`MetaDataModel.to_dict()`

## 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.

## hyperstream.models.plate module

**class** `hyperstream.models.plate.PlateDefinitionModel(*args, **values)`

Bases: `mongoengine.document.Document`

### **exception DoesNotExist**

Bases: `mongoengine.errors.DoesNotExist`

### **exception PlateDefinitionModel.MultipleObjectsReturned**

Bases: `mongoengine.errors.MultipleObjectsReturned`

### `PlateDefinitionModel.complement`

Boolean field type.

New in version 0.1.2.

### `PlateDefinitionModel.description`

A unicode string field.

### `PlateDefinitionModel.id`

A field wrapper around MongoDB's ObjectIds.

### `PlateDefinitionModel.meta_data_id`

A unicode string field.

### `PlateDefinitionModel.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.

### `PlateDefinitionModel.parent_plate`

A unicode string field.

PlateDefinitionModel.**plate\_id**

A unicode string field.

PlateDefinitionModel.**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 []

---

### hyperstream.models.stream module

**class** hyperstream.models.stream.**StreamDefinitionModel** (\*args, \*\*values)

Bases: mongoengine.document.Document

**exception DoesNotExist**

Bases: mongoengine.errors.DoesNotExist

**exception** StreamDefinitionModel.**MultipleObjectsReturned**

Bases: mongoengine.errors.MultipleObjectsReturned

StreamDefinitionModel.**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.

StreamDefinitionModel.**channel\_id**

A unicode string field.

StreamDefinitionModel.**get\_calculated\_intervals**()

StreamDefinitionModel.**id**

A field wrapper around MongoDB's ObjectIds.



`StreamDefinitionModel.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.

`StreamDefinitionModel.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.

`StreamDefinitionModel.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.

`StreamDefinitionModel.sandbox`

A unicode string field.

`StreamDefinitionModel.set_calculated_intervals` (*intervals*)

`StreamDefinitionModel.stream_id`

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

`StreamDefinitionModel.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** `StreamInstanceModel.MultipleObjectsReturned`  
Bases: `mongoengine.errors.MultipleObjectsReturned`

`StreamInstanceModel.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.

`StreamInstanceModel.id`  
A field wrapper around MongoDB's `ObjectIds`.

`StreamInstanceModel.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.

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

`StreamInstanceModel.stream_type`  
A unicode string field.

`StreamInstanceModel.value`  
A truly dynamic field type capable of handling different and varying types of data.  
Used by `DynamicDocument` to handle dynamic data

## 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.

## 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

## hyperstream.models.workflow module

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

**exception DoesNotExist**  
 Bases: `mongoengine.errors.DoesNotExist`

**exception WorkflowDefinitionModel.MultipleObjectsReturned**  
 Bases: `mongoengine.errors.MultipleObjectsReturned`

`WorkflowDefinitionModel.description`  
 A unicode string field.

`WorkflowDefinitionModel.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.

`WorkflowDefinitionModel.id`

A field wrapper around MongoDB's `ObjectIds`.

`WorkflowDefinitionModel.monitor`

Boolean field type.

New in version 0.1.2.

`WorkflowDefinitionModel.name`

A unicode string field.

`WorkflowDefinitionModel.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.

`WorkflowDefinitionModel.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.

`WorkflowDefinitionModel.online`

Boolean field type.

New in version 0.1.2.

`WorkflowDefinitionModel.owner`

A unicode string field.

`WorkflowDefinitionModel.workflow_id`

A unicode string field.

**class** `hyperstream.models.workflow.WorkflowStatusModel` (\*args, \*\*values)

Bases: `mongoengine.document.Document`

**exception** `DoesNotExist`

Bases: `mongoengine.errors.DoesNotExist`

**exception** `WorkflowStatusModel.MultipleObjectsReturned`

Bases: `mongoengine.errors.MultipleObjectsReturned`

`WorkflowStatusModel.id`

A field wrapper around MongoDB's `ObjectIds`.

`WorkflowStatusModel.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.

`WorkflowStatusModel.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.

`WorkflowStatusModel.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.

`WorkflowStatusModel.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.

`WorkflowStatusModel.workflow_id`

A unicode string field.

## Module contents

### hyperstream.stream package

#### Submodules

#### 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 tool

**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**

### 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*)

### 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*)

### 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

### 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*)

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

**iteritems** ()

**itertimestamps** ()

**itervalues** ()

**last** (*default=None*)

**tail** (*n*)

**timestamps** ()

**values** ()

### Module contents

## hyperstream.tool package

### Submodules

#### 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



## 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

## 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

## 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

## 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

## 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

## Module contents

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

## hyperstream.utils package

### Submodules

#### 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*)

## 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*)

## 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

## 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:

## hyperstream.utils.utils module

### Module contents

## hyperstream.workflow package

### Submodules

#### hyperstream.workflow.factor module

#### hyperstream.workflow.meta\_data\_manager module

#### hyperstream.workflow.node module

#### hyperstream.workflow.plate module

#### hyperstream.workflow.plate\_manager module

#### hyperstream.workflow.workflow module

Workflow and WorkflowMonitor definitions.

**class** `hyperstream.workflow.workflow.Workflow` (*workflow\_id, name, description, owner, on-line=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

## 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*)

## Module contents

## Submodules

### hyperstream.channel\_manager module

### 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**

## hyperstream.config module

HyperStream configuration module.

**class** hyperstream.config.**HyperStreamConfig**  
Bases: hyperstream.utils.containers.Printable

Wrapper around the hyperstream configuration file (hyperstream\_config.json)

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

## hyperstream.hyperstream module

Main HyperStream class

**class** hyperstream.hyperstream.**HyperStream** (*loglevel=20, file\_logger=True, console\_logger=True, mqtt\_logger=None*)  
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](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** (\*args, \*\*kws)

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

## 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** (\*args, \*\*kwargs)

Execute the engine - currently simple executes all workflows.

## 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.

## 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*)

**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

## hyperstream.version module

### Module contents



### Submodules

#### 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 ()
```

## tests.test\_time\_interval module

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

## tests.test\_tool\_channel module

### 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`



### h

- hyperstream, 33
- hyperstream.channels, 9
  - hyperstream.channels.assets\_channel, 3
  - hyperstream.channels.base\_channel, 4
  - hyperstream.channels.database\_channel, 5
  - hyperstream.channels.file\_channel, 6
  - hyperstream.channels.memory\_channel, 7
  - hyperstream.channels.module\_channel, 8
  - hyperstream.channels.tool\_channel, 8
- hyperstream.client, 29
- hyperstream.config, 30
- hyperstream.hyperstream, 30
- hyperstream.itertools2, 9
- hyperstream.itertools2.itertools2, 9
- hyperstream.models, 17
  - hyperstream.models.factor, 9
  - hyperstream.models.meta\_data, 10
  - hyperstream.models.node, 11
  - hyperstream.models.plate, 11
  - hyperstream.models.stream, 12
  - hyperstream.models.time\_interval, 14
  - hyperstream.models.tool, 15
  - hyperstream.models.workflow, 15
- hyperstream.online\_engine, 31
- hyperstream.plugin\_manager, 31
- hyperstream.stream, 20
- hyperstream.stream.stream, 17
- hyperstream.stream.stream\_collections, 19
  - hyperstream.stream.stream\_id, 19
  - hyperstream.stream.stream\_instance, 19
  - hyperstream.stream.stream\_view, 20
- hyperstream.time\_interval, 32
- hyperstream.tool, 23
  - hyperstream.tool.aggregate\_tool, 20
  - hyperstream.tool.base\_tool, 21
  - hyperstream.tool.multi\_output\_tool, 21
  - hyperstream.tool.plate\_creation\_tool, 22
  - hyperstream.tool.selector\_tool, 22
  - hyperstream.tool.tool, 23
- hyperstream.utils, 26
  - hyperstream.utils.decorators, 23
  - hyperstream.utils.errors, 24
  - hyperstream.utils.serialization, 25
  - hyperstream.utils.time\_utils, 25
- hyperstream.version, 33
- hyperstream.workflow, 29
  - hyperstream.workflow.workflow, 26
  - hyperstream.workflow.workflow\_manager, 28

### t

- tests, 36
  - tests.helpers, 35
  - tests.test\_time\_interval, 36



## A

- absolute() (hyperstream.time\_interval.RelativeTimeInterval method), 32
- add\_workflow() (hyperstream.hyperstream.HyperStream method), 30
- add\_workflow() (hyperstream.workflow.workflow\_manager.WorkflowManager method), 29
- AggregateTool (class in hyperstream.tool.aggregate\_tool), 20
- alignment\_node (hyperstream.models.factor.FactorDefinitionModel attribute), 9
- all\_time() (hyperstream.time\_interval.TimeInterval class method), 32
- any\_set() (in module hyperstream.itertools2.itertools2), 9
- append() (hyperstream.stream.stream\_collections.StreamInstanceCollection method), 19
- as\_dict() (hyperstream.stream.stream\_id.StreamId method), 19
- as\_dict() (hyperstream.stream.stream\_instance.StreamInstance method), 19
- as\_list() (hyperstream.stream.stream\_instance.StreamInstance method), 19
- as\_raw() (hyperstream.stream.stream\_id.StreamId method), 19
- assert\_all\_close() (in module tests.helpers), 35
- assert\_dict\_equal() (in module tests.helpers), 35
- AssetsChannel (class in hyperstream.stream.channels.assets\_channel), 3
- AssetStream (class in hyperstream.stream.stream), 17
- B**
- BaseChannel (class in hyperstream.stream.channels.base\_channel), 4
- BaseTool (class in hyperstream.tool.base\_tool), 21
- C**
- calculated\_intervals (hyperstream.models.stream.StreamDefinitionModel attribute), 12
- calculated\_intervals (hyperstream.stream.stream.AssetStream attribute), 17
- calculated\_intervals (hyperstream.stream.stream.DatabaseStream attribute), 18
- calculated\_intervals (hyperstream.stream.stream.Stream attribute), 18
- channel\_id (hyperstream.models.node.NodeDefinitionModel attribute), 11
- channel\_id (hyperstream.models.stream.StreamDefinitionModel attribute), 12
- ChannelAlreadyExistsError, 24
- ChannelNotFoundError, 24
- check\_calculation\_times() (hyperstream.channels.memory\_channel.MemoryChannel method), 7
- check\_input\_stream\_count() (in module hyperstream.utils.decorators), 23
- check\_multi\_output\_plate\_compatibility() (hyperstream.workflow.workflow.Workflow static method), 26
- check\_output\_format() (in module hyperstream.utils.decorators), 23
- check\_plate\_compatibility() (hyperstream.workflow.workflow.Workflow static method), 26
- check\_tool\_defined() (in module hyperstream.utils.decorators), 23
- clear\_sessions() (hyperstream.hyperstream.HyperStream method), 30
- Client (class in hyperstream.client), 29
- client (hyperstream.client.Client attribute), 29
- code\_pickler() (in module hyperstream.workflow.workflow\_manager), 29
- code\_unpickler() (in module hyperstream.workflow.workflow\_manager), 29
- commit\_all() (hyperstream.workflow.workflow\_manager.WorkflowManager

- method), 29
- commit\_workflow() (hyperstream.workflow.workflow\_manager.WorkflowManager method), 29
- complement (hyperstream.models.plate.PlateDefinitionModel attribute), 11
- complement (hyperstream.models.plate.PlateModel attribute), 12
- component() (hyperstream.stream.stream\_view.StreamView method), 20
- component\_filter() (hyperstream.stream.stream\_view.StreamView method), 20
- compress() (hyperstream.time\_interval.TimeIntervals method), 32
- ConfigurationError, 24
- connect() (hyperstream.client.Client method), 29
- construct\_experiment\_id() (in module hyperstream.utils.time\_utils), 25
- count() (in module hyperstream.itertools2.itertools2), 9
- create\_factor() (hyperstream.workflow.workflow.Workflow method), 26
- create\_factor\_general() (hyperstream.workflow.workflow.Workflow method), 27
- create\_multi\_output\_factor() (hyperstream.workflow.workflow.Workflow method), 27
- create\_node() (hyperstream.workflow.workflow.Workflow method), 27
- create\_node\_creation\_factor() (hyperstream.workflow.workflow.Workflow method), 27
- create\_plate() (in module tests.helpers), 35
- create\_stream() (hyperstream.channels.assets\_channel.AssetsChannel method), 3
- create\_stream() (hyperstream.channels.base\_channel.BaseChannel method), 4
- create\_stream() (hyperstream.channels.database\_channel.DatabaseChannel method), 5
- create\_stream() (hyperstream.channels.memory\_channel.MemoryChannel method), 7
- create\_stream() (hyperstream.channels.memory\_channel.ReadOnlyMemoryChannel method), 8
- create\_workflow() (hyperstream.hyperstream.HyperStream method), 30
- current\_session (hyperstream.hyperstream.HyperStream attribute), 31
- D**
- data (hyperstream.models.meta\_data.MetaDataModel attribute), 10
- data\_loader() (hyperstream.channels.file\_channel.FileChannel method), 6
- data\_loader() (hyperstream.channels.module\_channel.ModuleChannel method), 8
- DatabaseChannel (class in hyperstream.channels.database\_channel), 5
- DatabaseStream (class in hyperstream.stream.stream), 17
- datetime (hyperstream.models.stream.StreamInstanceModel attribute), 14
- datetime2unix() (in module hyperstream.utils.time\_utils), 25
- db (hyperstream.client.Client attribute), 29
- delete\_meta\_data() (in module tests.helpers), 35
- delete\_nones() (hyperstream.stream.stream\_view.StreamView method), 20
- delete\_plate() (in module tests.helpers), 35
- delete\_workflow() (hyperstream.workflow.workflow\_manager.WorkflowManager method), 29
- description (hyperstream.models.factor.OutputPlateDefinitionModel attribute), 10
- description (hyperstream.models.plate.PlateDefinitionModel attribute), 11
- description (hyperstream.models.workflow.WorkflowDefinitionModel attribute), 15
- dict\_items() (hyperstream.stream.stream\_view.StreamView method), 20
- dict\_iteritems() (hyperstream.stream.stream\_view.StreamView method), 20
- duration2str() (in module hyperstream.utils.time\_utils), 25
- E**
- end (hyperstream.models.time\_interval.TimeIntervalModel attribute), 14
- end (hyperstream.time\_interval.RelativeTimeInterval attribute), 32
- end (hyperstream.time\_interval.TimeInterval attribute), 32
- end (hyperstream.time\_interval.TimeIntervals attribute), 32
- execute() (hyperstream.online\_engine.OnlineEngine method), 31
- execute() (hyperstream.tool.multi\_output\_tool.MultiOutputTool method), 22
- execute() (hyperstream.tool.plate\_creation\_tool.PlateCreationTool method), 22
- execute() (hyperstream.tool.selector\_tool.SelectorTool method), 22
- execute() (hyperstream.tool.tool.Tool method), 23

- execute() (hyperstream.workflow.workflow.Workflow method), 28
- execute\_all() (hyperstream.workflow.workflow\_manager.WorkflowManager method), 29
- execute\_tool() (hyperstream.channels.base\_channel.BaseChannel method), 4
- extend() (hyperstream.stream.stream\_collections.StreamInstanceCollection method), 19
- ## F
- factor\_type (hyperstream.models.factor.FactorDefinitionModel attribute), 9
- FactorAlreadyExistsError, 24
- FactorDefinitionError, 24
- FactorDefinitionModel (class in hyperstream.models.factor), 9
- factorgraph\_viz() (hyperstream.workflow.workflow.Workflow static method), 28
- factors (hyperstream.models.workflow.WorkflowDefinitionModel attribute), 15
- file\_filter() (hyperstream.channels.file\_channel.FileChannel method), 7
- file\_filter() (hyperstream.channels.module\_channel.ModuleChannel method), 8
- FileChannel (class in hyperstream.channels.file\_channel), 6
- FileDateTimeVersion (class in hyperstream.channels.file\_channel), 7
- find\_stream() (hyperstream.channels.base\_channel.BaseChannel method), 4
- find\_streams() (hyperstream.channels.base\_channel.BaseChannel method), 4
- first() (hyperstream.stream.stream\_view.StreamView method), 20
- func\_dump() (in module hyperstream.utils.serialization), 25
- func\_load() (in module hyperstream.utils.serialization), 25
- func\_reconstruct\_closure() (in module hyperstream.utils.serialization), 25
- ## G
- get\_calculated\_intervals() (hyperstream.models.stream.StreamDefinitionModel method), 12
- get\_config\_value() (hyperstream.client.Client method), 30
- get\_meta\_data() (in module tests.helpers), 35
- get\_model() (hyperstream.tool.base\_tool.BaseTool method), 21
- get\_or\_create\_stream() (hyperstream.channels.base\_channel.BaseChannel method), 4
- get\_results() (hyperstream.channels.base\_channel.BaseChannel method), 5
- get\_results() (hyperstream.channels.database\_channel.DatabaseChannel method), 6
- get\_results() (hyperstream.channels.file\_channel.FileChannel method), 7
- get\_results() (hyperstream.channels.memory\_channel.MemoryChannel method), 7
- get\_results() (hyperstream.channels.memory\_channel.ReadOnlyMemoryChannel method), 8
- get\_results() (hyperstream.channels.tool\_channel.ToolChannel method), 8
- get\_stream\_id() (in module hyperstream.stream.stream\_id), 19
- get\_stream\_writer() (hyperstream.channels.base\_channel.BaseChannel method), 5
- get\_stream\_writer() (hyperstream.channels.database\_channel.DatabaseChannel method), 6
- get\_stream\_writer() (hyperstream.channels.memory\_channel.MemoryChannel method), 7
- get\_stream\_writer() (hyperstream.channels.memory\_channel.ReadOnlyMemoryChannel method), 8
- get\_timedelta() (in module hyperstream.utils.time\_utils), 25
- ## H
- handle\_exception() (in module hyperstream.utils.errors), 25
- head() (hyperstream.stream.stream\_view.StreamView method), 20
- humanized (hyperstream.time\_interval.TimeInterval attribute), 32
- humanized (hyperstream.time\_interval.TimeIntervals attribute), 32
- HyperStream (class in hyperstream.hyperstream), 30
- hyperstream (module), 33
- hyperstream.channels (module), 9
- hyperstream.channels.assets\_channel (module), 3
- hyperstream.channels.base\_channel (module), 4
- hyperstream.channels.database\_channel (module), 5
- hyperstream.channels.file\_channel (module), 6
- hyperstream.channels.memory\_channel (module), 7
- hyperstream.channels.module\_channel (module), 8
- hyperstream.channels.tool\_channel (module), 8
- hyperstream.client (module), 29
- hyperstream.config (module), 30
- hyperstream.hyperstream (module), 30
- hyperstream.itertools2 (module), 9
- hyperstream.itertools2.itertools2 (module), 9
- hyperstream.models (module), 17

- hyperstream.models.factor (module), 9
  - hyperstream.models.meta\_data (module), 10
  - hyperstream.models.node (module), 11
  - hyperstream.models.plate (module), 11
  - hyperstream.models.stream (module), 12
  - hyperstream.models.time\_interval (module), 14
  - hyperstream.models.tool (module), 15
  - hyperstream.models.workflow (module), 15
  - hyperstream.online\_engine (module), 31
  - hyperstream.plugin\_manager (module), 31
  - hyperstream.stream (module), 20
  - hyperstream.stream.stream (module), 17
  - hyperstream.stream.stream\_collections (module), 19
  - hyperstream.stream.stream\_id (module), 19
  - hyperstream.stream.stream\_instance (module), 19
  - hyperstream.stream.stream\_view (module), 20
  - hyperstream.time\_interval (module), 32
  - hyperstream.tool (module), 23
  - hyperstream.tool.aggregate\_tool (module), 20
  - hyperstream.tool.base\_tool (module), 21
  - hyperstream.tool.multi\_output\_tool (module), 21
  - hyperstream.tool.plate\_creation\_tool (module), 22
  - hyperstream.tool.selector\_tool (module), 22
  - hyperstream.tool.tool (module), 23
  - hyperstream.utils (module), 26
  - hyperstream.utils.decorators (module), 23
  - hyperstream.utils.errors (module), 24
  - hyperstream.utils.serialization (module), 25
  - hyperstream.utils.time\_utils (module), 25
  - hyperstream.version (module), 33
  - hyperstream.workflow (module), 29
  - hyperstream.workflow.workflow (module), 26
  - hyperstream.workflow.workflow\_manager (module), 28
  - HyperStreamConfig (class in hyperstream.config), 30
  - HyperStreamTimeIntervalTests (class in tests.test\_time\_interval), 36
- I**
- id (hyperstream.models.meta\_data.MetaDataModel attribute), 10
  - id (hyperstream.models.plate.PlateDefinitionModel attribute), 11
  - id (hyperstream.models.stream.StreamDefinitionModel attribute), 12
  - id (hyperstream.models.stream.StreamInstanceModel attribute), 14
  - id (hyperstream.models.workflow.WorkflowDefinitionModel attribute), 16
  - id (hyperstream.models.workflow.WorkflowStatusModel attribute), 16
  - identifier (hyperstream.models.meta\_data.MetaDataModel attribute), 10
  - IncompatiblePlatesError, 24
  - IncompatibleToolError, 24
- insert\_meta\_data() (in module tests.helpers), 35
  - is\_close() (in module tests.helpers), 35
  - is\_empty (hyperstream.time\_interval.TimeIntervals attribute), 32
  - is\_function (hyperstream.models.tool.ToolParameterModel attribute), 15
  - is\_naive() (in module hyperstream.utils.time\_utils), 25
  - is\_python (hyperstream.channels.file\_channel.FileDateTimeVersion attribute), 7
  - is\_set (hyperstream.models.tool.ToolParameterModel attribute), 15
  - items() (hyperstream.stream.stream\_view.StreamView method), 20
  - iteritems() (hyperstream.stream.stream\_view.StreamView method), 20
  - itertimestamps() (hyperstream.stream.stream\_view.StreamView method), 20
  - itervalues() (hyperstream.stream.stream\_view.StreamView method), 20
- J**
- json\_serial() (in module hyperstream.utils.time\_utils), 25
- K**
- key (hyperstream.models.tool.ToolParameterModel attribute), 15
- L**
- last() (hyperstream.stream.stream\_view.StreamView method), 20
  - last\_accessed (hyperstream.models.stream.StreamDefinitionModel attribute), 12
  - last\_accessed (hyperstream.models.workflow.WorkflowStatusModel attribute), 16
  - last\_accessed (hyperstream.stream.stream.DatabaseStream attribute), 18
  - last\_messages (tests.helpers.MqttClient attribute), 35
  - last\_updated (hyperstream.models.stream.StreamDefinitionModel attribute), 13
  - last\_updated (hyperstream.models.workflow.WorkflowStatusModel attribute), 17
  - last\_updated (hyperstream.stream.stream.DatabaseStream attribute), 18
  - LinkageError, 24
  - load() (hyperstream.stream.stream.DatabaseStream method), 18
  - load\_channels() (hyperstream.plugin\_manager.Plugin method), 31
  - load\_workflow() (hyperstream.workflow.workflow\_manager.WorkflowManager method), 29



## M

- MemoryChannel (class in hyperstream.channels.memory\_channel), 7
- message (hyperstream.utils.errors.FactorAlreadyExistsError attribute), 24
- message (hyperstream.utils.errors.NodeAlreadyExistsError attribute), 24
- message (hyperstream.utils.errors.PlateDefinitionError attribute), 24
- message (hyperstream.utils.errors.PlateEmptyError attribute), 24
- message (hyperstream.utils.errors.StreamNotAvailableError attribute), 24
- message (hyperstream.utils.errors.ToolExecutionError attribute), 25
- message() (hyperstream.tool.base\_tool.BaseTool method), 21
- meta\_data (hyperstream.models.stream.StreamIdField attribute), 13
- meta\_data\_id (hyperstream.models.factor.OutputPlateDefinitionModel attribute), 10
- meta\_data\_id (hyperstream.models.plate.PlateDefinitionModel attribute), 11
- meta\_data\_id (hyperstream.models.plate.PlateModel attribute), 12
- MetaDataModel (class in hyperstream.models.meta\_data), 10
- MetaDataModel.DoesNotExist, 10
- MetaDataModel.MultipleObjectsReturned, 10
- ModuleChannel (class in hyperstream.channels.module\_channel), 8
- monitor (hyperstream.models.workflow.WorkflowDefinitionModel attribute), 16
- mosquito\_is\_running() (in module tests.helpers), 35
- MqttClient (class in tests.helpers), 35
- MultiOutputTool (class in hyperstream.tool.multi\_output\_tool), 21
- MultipleStreamsFoundError, 24
- non\_empty\_streams (hyperstream.channels.memory\_channel.MemoryChannel attribute), 7
- now\_minus() (hyperstream.time\_interval.TimeInterval class method), 32

## O

- objects (hyperstream.models.meta\_data.MetaDataModel attribute), 10
- objects (hyperstream.models.plate.PlateDefinitionModel attribute), 11
- objects (hyperstream.models.stream.StreamDefinitionModel attribute), 13
- objects (hyperstream.models.stream.StreamInstanceModel attribute), 14
- objects (hyperstream.models.workflow.WorkflowDefinitionModel attribute), 16
- objects (hyperstream.models.workflow.WorkflowStatusModel attribute), 17
- on\_message() (tests.helpers.MqttClient method), 35
- on\_message() (tests.helpers.MqttClient method), 35
- online (hyperstream.models.workflow.WorkflowDefinitionModel attribute), 16
- online\_average() (in module hyperstream.itertools2.itertools2), 9
- online\_product() (in module hyperstream.itertools2.itertools2), 9
- online\_sum() (in module hyperstream.itertools2.itertools2), 9
- online\_variance() (in module hyperstream.itertools2.itertools2), 9
- OnlineEngine (class in hyperstream.online\_engine), 31
- OnlineEngineConfig (class in hyperstream.config), 30
- output\_plate (hyperstream.models.factor.FactorDefinitionModel attribute), 9
- OutputPlateDefinitionModel (class in hyperstream.models.factor), 10
- owner (hyperstream.models.workflow.WorkflowDefinitionModel attribute), 16

## N

- name (hyperstream.models.stream.StreamIdField attribute), 13
- name (hyperstream.models.tool.ToolModel attribute), 15
- name (hyperstream.models.workflow.WorkflowDefinitionModel attribute), 16
- name (hyperstream.tool.base\_tool.BaseTool attribute), 21
- new\_session() (hyperstream.hyperstream.HyperStream method), 31
- NodeAlreadyExistsError, 24
- NodeDefinitionError, 24
- NodeDefinitionModel (class in hyperstream.models.node), 11
- nodes (hyperstream.models.workflow.WorkflowDefinitionModel attribute), 16

## P

- parameters (hyperstream.models.tool.ToolModel attribute), 15
- parameters (hyperstream.tool.base\_tool.BaseTool attribute), 21
- parameters\_dict (hyperstream.tool.base\_tool.BaseTool attribute), 21
- parameters\_from\_dicts() (hyperstream.tool.base\_tool.BaseTool static method), 21
- parameters\_from\_model() (hyperstream.tool.base\_tool.BaseTool static method), 21

- parent (hyperstream.models.meta\_data.MetaDataModel attribute), 10
- parent\_node (hyperstream.stream.stream.Stream attribute), 18
- parent\_plate (hyperstream.models.plate.PlateDefinitionModel attribute), 11
- parse() (hyperstream.time\_interval.TimeIntervals method), 32
- parse\_time\_tuple() (in module hyperstream.time\_interval), 32
- path (hyperstream.channels.file\_channel.FileChannel attribute), 7
- plate\_id (hyperstream.models.factor.OutputPlateDefinitionModel attribute), 10
- plate\_id (hyperstream.models.plate.PlateDefinitionModel attribute), 11
- plate\_ids (hyperstream.models.node.NodeDefinitionModel attribute), 11
- PlateCreationTool (class in hyperstream.tool.plate\_creation\_tool), 22
- PlateDefinitionError, 24
- PlateDefinitionModel (class in hyperstream.models.plate), 11
- PlateDefinitionModel.DoesNotExist, 11
- PlateDefinitionModel.MultipleObjectsReturned, 11
- PlateEmptyError, 24
- PlateModel (class in hyperstream.models.plate), 12
- PlateNotFoundError, 24
- Plugin (class in hyperstream.plugin\_manager), 31
- populate\_tools\_and\_factors() (hyperstream.hyperstream.HyperStream method), 31
- profile() (in module hyperstream.time\_interval), 33
- purge() (hyperstream.stream.stream.Stream method), 18
- purge\_all() (hyperstream.channels.memory\_channel.MemoryChannel method), 7
- purge\_node() (hyperstream.channels.base\_channel.BaseChannel method), 5
- purge\_stream() (hyperstream.channels.assets\_channel.AssetsChannel method), 3
- purge\_stream() (hyperstream.channels.base\_channel.BaseChannel method), 5
- purge\_stream() (hyperstream.channels.database\_channel.DatabaseChannel method), 6
- purge\_stream() (hyperstream.channels.memory\_channel.MemoryChannel method), 7
- ## R
- ReadOnlyMemoryChannel (class in hyperstream.channels.memory\_channel), 7
- reconstruct\_interval() (in module hyperstream.utils.time\_utils), 25
- RelativeTimeInterval (class in hyperstream.time\_interval), 32
- remove\_microseconds() (in module hyperstream.utils.time\_utils), 25
- requested\_intervals (hyperstream.models.workflow.WorkflowStatusModel attribute), 17
- requested\_intervals (hyperstream.workflow.workflow.Workflow attribute), 28
- resource\_manager() (in module tests.helpers), 35
- ## S
- sandbox (hyperstream.models.stream.StreamDefinitionModel attribute), 13
- save() (hyperstream.stream.stream.DatabaseStream method), 18
- SelectorTool (class in hyperstream.tool.selector\_tool), 22
- session (hyperstream.client.Client attribute), 30
- sessions (hyperstream.hyperstream.HyperStream attribute), 31
- set\_all\_requested\_intervals() (hyperstream.workflow.workflow\_manager.WorkflowManager method), 29
- set\_calculated\_intervals() (hyperstream.models.stream.StreamDefinitionModel method), 13
- set\_requested\_intervals() (hyperstream.workflow.workflow\_manager.WorkflowManager method), 29
- set\_tool\_reference() (hyperstream.stream.stream.Stream method), 18
- setup() (in module tests.helpers), 35
- sinks (hyperstream.models.factor.FactorDefinitionModel attribute), 9
- sources (hyperstream.models.factor.FactorDefinitionModel attribute), 9
- span (hyperstream.time\_interval.TimeIntervals attribute), 32
- split() (hyperstream.time\_interval.TimeIntervals method), 32
- splitting\_node (hyperstream.models.factor.FactorDefinitionModel attribute), 10
- start (hyperstream.models.time\_interval.TimeIntervalModel attribute), 14
- start (hyperstream.time\_interval.RelativeTimeInterval attribute), 32
- start (hyperstream.time\_interval.TimeInterval attribute), 32
- start (hyperstream.time\_interval.TimeIntervals attribute), 32
- Stream (class in hyperstream.stream.stream), 18
- stream\_id (hyperstream.models.stream.StreamDefinitionModel attribute), 13
- stream\_id (hyperstream.models.stream.StreamInstanceModel attribute), 14

- stream\_name (hyperstream.models.node.NodeDefinitionModel attribute), 11
- stream\_type (hyperstream.models.stream.StreamDefinitionModel attribute), 13
- stream\_type (hyperstream.models.stream.StreamInstanceModel attribute), 14
- StreamAlreadyExistsError, 24
- StreamDataNotAvailableError, 24
- StreamDefinitionModel (class in hyperstream.models.stream), 12
- StreamDefinitionModel.DoesNotExist, 12
- StreamDefinitionModel.MultipleObjectsReturned, 12
- StreamDict (class in hyperstream.stream.stream\_collections), 19
- StreamId (class in hyperstream.stream.stream\_id), 19
- StreamIdField (class in hyperstream.models.stream), 13
- StreamInstance (class in hyperstream.stream.stream\_instance), 19
- StreamInstanceCollection (class in hyperstream.stream.stream\_collections), 19
- StreamInstanceModel (class in hyperstream.models.stream), 13
- StreamInstanceModel.DoesNotExist, 13
- StreamInstanceModel.MultipleObjectsReturned, 13
- StreamMetaInstance (class in hyperstream.stream.stream\_instance), 19
- StreamNotAvailableError, 24
- StreamNotFoundError, 24
- StreamView (class in hyperstream.stream.stream\_view), 20
- ## T
- tag (hyperstream.models.meta\_data.MetaDataModel attribute), 10
- tail() (hyperstream.stream.stream\_view.StreamView method), 20
- teardown() (in module tests.helpers), 35
- test\_constructors() (tests.test\_time\_interval.HyperStreamTimeIntervalTests method), 36
- test\_relative\_time\_interval() (tests.test\_time\_interval.HyperStreamTimeIntervalTests method), 36
- test\_time\_interval() (tests.test\_time\_interval.HyperStreamTimeIntervalTests method), 36
- tests (module), 36
- tests.helpers (module), 35
- tests.test\_time\_interval (module), 36
- TimeInterval (class in hyperstream.time\_interval), 32
- TimeIntervalModel (class in hyperstream.models.time\_interval), 14
- TimeIntervals (class in hyperstream.time\_interval), 32
- timeit() (in module hyperstream.utils.decorators), 23
- timestamps() (hyperstream.stream.stream\_view.StreamView method), 20
- to\_dict() (hyperstream.models.meta\_data.MetaDataModel method), 10
- to\_dict() (hyperstream.workflow.workflow.Workflow method), 28
- to\_json() (hyperstream.stream.stream\_id.StreamId method), 19
- to\_json() (hyperstream.time\_interval.TimeInterval method), 32
- to\_json() (hyperstream.time\_interval.TimeIntervals method), 32
- to\_json() (hyperstream.workflow.workflow.Workflow method), 28
- to\_tuple() (hyperstream.time\_interval.TimeInterval method), 32
- Tool (class in hyperstream.tool.tool), 23
- tool (hyperstream.models.factor.FactorDefinitionModel attribute), 10
- ToolChannel (class in hyperstream.channels.tool\_channel), 8
- ToolExecutionError, 24
- ToolInitialisationError, 25
- ToolModel (class in hyperstream.models.tool), 15
- ToolNotFoundError, 25
- ToolParameterModel (class in hyperstream.models.tool), 15
- ## U
- unix2datetime() (in module hyperstream.utils.time\_utils), 25
- up\_to\_now() (hyperstream.time\_interval.TimeInterval class method), 32
- update\_state() (hyperstream.channels.memory\_channel.ReadOnlyMemoryChannel method), 8
- update\_state() (hyperstream.channels.module\_channel.ModuleChannel method), 8
- update\_streams() (hyperstream.channels.assets\_channel.AssetsChannel method), 4
- update\_streams() (hyperstream.channels.base\_channel.BaseChannel method), 5
- update\_streams() (hyperstream.channels.database\_channel.DatabaseChannel method), 6
- update\_streams() (hyperstream.channels.file\_channel.FileChannel method), 7
- update\_streams() (hyperstream.channels.memory\_channel.MemoryChannel method), 7
- update\_streams() (hyperstream.channels.memory\_channel.ReadOnlyMemoryChannel method), 8

`use_provided_values` (hyperstream.models.factor.OutputPlateDefinitionModel attribute), 10

`utcnow()` (in module `hyperstream.utils.time_utils`), 25

## V

`value` (hyperstream.models.stream.StreamInstanceModel attribute), 14

`value` (hyperstream.models.tool.ToolParameterModel attribute), 15

`values` (hyperstream.models.plate.PlateDefinitionModel attribute), 12

`values` (hyperstream.models.plate.PlateModel attribute), 12

`values()` (hyperstream.stream.stream\_view.StreamView method), 20

`version` (hyperstream.models.tool.ToolModel attribute), 15

`versions` (hyperstream.channels.module\_channel.ModuleChannel attribute), 8

## W

`walk()` (hyperstream.channels.file\_channel.FileChannel static method), 7

`width` (hyperstream.time\_interval.TimeInterval attribute), 32

`window()` (hyperstream.stream.stream.Stream method), 18

`Workflow` (class in `hyperstream.workflow.workflow`), 26

`workflow_id` (hyperstream.models.workflow.WorkflowDefinitionModel attribute), 16

`workflow_id` (hyperstream.models.workflow.WorkflowStatusModel attribute), 17

`WorkflowDefinitionModel` (class in `hyperstream.models.workflow`), 15

`WorkflowDefinitionModel.DoesNotExist`, 15

`WorkflowDefinitionModel.MultipleObjectsReturned`, 15

`WorkflowManager` (class in `hyperstream.workflow.workflow_manager`), 28

`WorkflowMonitor` (class in `hyperstream.workflow.workflow`), 28

`WorkflowStatusModel` (class in `hyperstream.models.workflow`), 16

`WorkflowStatusModel.DoesNotExist`, 16

`WorkflowStatusModel.MultipleObjectsReturned`, 16

`write_to_history()` (hyperstream.tool.base\_tool.BaseTool static method), 21

`write_to_stream()` (hyperstream.channels.assets\_channel.AssetsChannel method), 4

`writer` (hyperstream.stream.stream.Stream attribute), 19