

---

# StreamPy Documentation

*Release*

**Author**

June 15, 2016



<b>1</b>	<b>Agent module</b>	<b>3</b>
<b>2</b>	<b>ML package</b>	<b>7</b>
2.1	Subpackages . . . . .	7
2.1.1	ML.Geomap package . . . . .	7
	Submodules . . . . .	7
	ML.Geomap.geomap module . . . . .	7
	Module contents . . . . .	7
2.1.2	ML.KMeans package . . . . .	7
	Submodules . . . . .	7
	ML.KMeans.KMeansStream module . . . . .	7
	ML.KMeans.kmeans module . . . . .	8
	Module contents . . . . .	11
2.1.3	ML.LinearRegression package . . . . .	11
	Submodules . . . . .	11
	ML.LinearRegression.LinearRegressionStream module . . . . .	11
	ML.LinearRegression.linear_regression module . . . . .	12
	Module contents . . . . .	14
2.2	Submodules . . . . .	14
2.3	ML.Stream_Learn module . . . . .	14
2.4	Module contents . . . . .	16
<b>3</b>	<b>MakeProcess module</b>	<b>17</b>
<b>4</b>	<b>Operators module</b>	<b>19</b>
<b>5</b>	<b>RemoteQueue module</b>	<b>23</b>
<b>6</b>	<b>Stream module</b>	<b>25</b>
<b>7</b>	<b>SystemParameters module</b>	<b>33</b>
<b>8</b>	<b>examples_element_wrapper module</b>	<b>35</b>
<b>9</b>	<b>examples_timed_window_wrapper module</b>	<b>39</b>
<b>10</b>	<b>examples_window_wrapper module</b>	<b>41</b>
<b>11</b>	<b>source_stream module</b>	<b>43</b>

<b>12 tutorial_average_of_stream module</b>	<b>45</b>
<b>13 tutorial_element_wrapper_stream_operator module</b>	<b>47</b>
<b>14 tutorial_exponential_smoothing module</b>	<b>49</b>
<b>15 tutorial_filter_1 module</b>	<b>51</b>
<b>16 tutorial_many_to_many module</b>	<b>53</b>
<b>17 tutorial_merge module</b>	<b>55</b>
<b>18 tutorial_sine module</b>	<b>57</b>
<b>19 tutorial_split module</b>	<b>59</b>
<b>20 weave_streams module</b>	<b>61</b>
<b>21 Indices and tables</b>	<b>63</b>
<b>Python Module Index</b>	<b>65</b>

Contents:



---

## Agent module

---

This module contains the Agent class. The Agent and Stream classes are the building blocks of PythonStreams.

```
class Agent.Agent (in_streams, out_streams, transition, state=None, call_streams=None,  
                  stream_manager=None, name=None)
```

Bases: object

An agent is an automaton: a state-transition machine. An agent is initialized in `__init__` and a state transition is executed by `next()`.

An agent has lists of: (1) input streams, (2) output streams and (3) call streams. Streams are described in `Stream.py`.

During a state transition an agent: (1) May read values from its input streams. (Note that

reading values in a stream does not change the stream.)

2.Append values to the tails of its output streams.

3.Change the agent's own state.

When a call stream is modified the agent's `next()` method is called which causes the agent to execute a state transition.

The default is that every input stream is also a call stream, i.e., the agent executes a state transition when any of its input streams is modified. For performance reasons, we may not want the agent to execute state transitions each time any of its input streams is modified; we may want the agent to execute state transitions periodically — for example, every second. In this case, the call streams will be different from the input streams. A call stream that has a value appended to it every second will cause the agent to execute a state transition every second.

**Parameters** `in_streams` : list of streams

**out\_streams** : list of streams

**call\_streams** : list of streams

When a new value is added to a stream in this list a state transition is invoked. This the usual way (but not the only way) in which state transitions occur. A state transition for an agent `ag` can also be executed by calling `ag.next()`

**state:** object

The state of the agent. The state is updated after a transition.

**transition:** function

This function is called by `next()` which is the state-transition operation for this agent. An agent's state transition is specified by its transition function.

**stream\_manager** : function

Each stream has management variables, such as whether the stream is open or closed. After a state-transition the agent executes the stream\_manager function to modify the management variables of the agent's output and call streams.

**name** : str, optional

name of this agent

### Attributes

<b>_in_lists</b> : list of InList	InList defines the slice of a list. The j-th element of _in_lists is an InList that defines the slice of the j-th input stream that may be read by this agent in a state transition. For example, if <code>listj = _in_lists[j].lists</code> <code>startj = _in_lists[j].start</code> <code>stopj = _in_lists[j].stop</code> Then this agent can read the slice: <code>listj[startj:stopj]</code> of the jth input stream. This slice is a slice of the most recent values of the stream.
<b>_out_lists</b> : list	The j-th element of _out_lists is the list of values to be appended to the j-th output stream after the state transition.

### Methods

<b>next</b> (stream_name=None)	Execute a state transition. The method has 3 parts: (i) set up the data structures to execute a state transition, (ii) call transition to: (a) get the values to be appended to output streams, (b) get the next state, and (c) update 'start' indices for each input stream. The 'start' pointers are indices where the agent asserts that it will no longer access elements of its input streams with indices earlier (i.e. smaller) than 'start'. (iii) update data structures after the transition.
--------------------------------	---

**next** (stream\_name=None)

Execute the next state transition.

This function does the following: Part 1: set up data structures for the state transition. Part 2: execute the state transition by calling self.transition Part 3: update data structures after the transition.

This method can be called by any agent and is called whenever a value is appended to any stream in call\_streams

**Parameters** **stream\_name** : str, optional

A new value was appended to the stream with name stream\_name as a result of which this agent executes a state transition.

Agent.**EPSILON** = 1e-12

**class** Agent.**InList**

Bases: tuple

### Attributes

<i>list</i>	Alias for field number 0
<i>start</i>	Alias for field number 1
<i>stop</i>	Alias for field number 2



## Methods

---

`count(...)`

---

`index((value, [start, ...)` Raises `ValueError` if the value is not present.

---

### **list**

Alias for field number 0

### **start**

Alias for field number 1

### **stop**

Alias for field number 2



## 2.1 Subpackages

### 2.1.1 ML.Geomap package

#### Submodules

#### ML.Geomap.geomap module

#### Module contents

### 2.1.2 ML.KMeans package

#### Submodules

#### ML.KMeans.KMeansStream module

**class** `ML.KMeans.KMeansStream.KMeansStream` (*draw, output, k, incremental=True, figsize=(15, 8)*)  
Helper class for kmeans clustering.

This class provides train and predict functions for using kmeans with *Stream\_Learn*.

**Parameters** **draw** : boolean

Describes whether the data is to be plotted (data must have 2 or less dimensions).

**output** : boolean

Describes whether debug info is to be printed. Info includes average error, average number of iterations, current number of iterations, and number of changed points over time.

**k** : int

Describes the number of clusters to train.

**incremental** : boolean, optional

Describes whether the kmeans algorithm is run incrementally or not (the default is True). If incremental, then previous clusters are used to initialize new clusters. Otherwise, clusters are reinitialized randomly for each window.

**figsize** : tuple, optional

A tuple containing the width and height of the plot for the map (the default is (15, 8)).

### Attributes

train	(function) The train function with signature as required by <i>Stream_Learn</i> .
predict	(function) The predict function with signature as required by 'Stream_Learn'.
avg_iterations	(float) The average number of iterations per window of data trained.
avg_error	(float) The average error per window of data trained.

### Methods

---

<code>reset()</code>	Resets the KMeans functions and average values.
----------------------	---

---

#### **reset ( )**

Resets the KMeans functions and average values.

Resets: train, predict, avg\_iterations, avg\_error

## ML.KMeans.kmeans module

`ML.KMeans.kmeans.computeCentroids` (*X*, *index*, *k*)

Finds the centroids for the data given the index of the closest centroid for each data point.

**Parameters** *X* : numpy.ndarray

A numpy array with dimensions  $n * 2$  for some integer  $n$ .

**index** : numpy.ndarray

A numpy array with dimensions  $n * 1$  that describes the closest centroid to each point in *X*.

**k** : int

Describes the number of centroids.  $k - 1$  is the maximum value that appears in *index*.

**Returns** *centroids* : numpy.ndarray

A numpy array with dimensions  $k * 2$ .

### Notes

The centroids are computed by taking the mean of each group of points in *X* with the same index value. For  $i$  in  $[0, k)$ , *centroids*[*i*] is the mean of all data points *X*[*j*] where *index*[*j*] is *i*.

`ML.KMeans.kmeans.evaluate_error` (*X*, *centroids*, *index*)

Returns the mean squared error.

**Parameters** *X* : numpy.ndarray

A numpy array with 2 columns.

**centroids** : numpy.ndarray

A numpy array with 2 columns.

**index** : numpy.ndarray

A numpy array with 1 column.

**Returns** float

The mean squared error.

## Notes

The mean squared error is calculated as the average squared distance of each point from the closest centroid.

`ML.KMeans.kmeans.findClosestCentroids(X, centroids)`

Returns a numpy array containing the index of the closest centroid for each point in *X*.

**Parameters** *X* : numpy.ndarray

A numpy array with 2 columns.

**centroids** : numpy.ndarray

A numpy array with 2 columns.

**Returns** **index** : numpy.ndarray

A numpy array with dimensions  $n * 1$ , where  $n$  is the number of rows in *X*. For each row  $i$  in *index*, *index*[ $i$ ] is in  $[0, k)$  where  $k$  is the number of rows in *centroids*.

`ML.KMeans.kmeans.init_plot(figsize=(15, 8))`

Initializes the plot.

**Parameters** **figsize** : tuple, optional

A tuple containing the width and height of the plot (the default is (15, 8)).

`ML.KMeans.kmeans.initialize(k, low, high)`

Returns  $k$  random points with  $x$  and  $y$  coordinates in  $[low, high)$ .

**Parameters** **k** : int

The number of points to return.

**low** : int

The lower bound (inclusive) for a point.

**high** : int

The upper bound (exclusive) for a point.

**Returns** **centroids** : numpy.ndarray

Numpy array with dimensions  $k$  by 2.

`ML.KMeans.kmeans.initializeCentroids(X, k)`

Returns  $k$  random points from the data *X* without replacement.

**Parameters** *X* : numpy.ndarray

A numpy array with dimensions  $n * 2$ , where  $n \geq k$ .

**k** : int

The number of points to return

**Returns** numpy.ndarray

Numpy array with dimensions  $k$  by 2.

`ML.KMeans.kmeans.initializeData` ( $n, k, scale, low, high$ )

Initialize  $n$  points around  $k$  random centroids each with a normal distribution and scale.

**Parameters**  $n$  : int

Describes the numbe of points to make around each centroid.

$k$  : int

Describes the number of centroids.

$scale$  : int

Describes the scale for the distribution.

$low$  : int

The lower bound (inclusive) for a centroid.

$high$  : int

The upper bound (exclusive) for a centroid.

**Returns**  $X$  : numpy.ndarray

A numpy array with dimensions  $(n * k) * 2$ .

`ML.KMeans.kmeans.initializeDataCenter` ( $centroid, scale, n$ )

Initialize  $n$  points with a normal distribution and scale around a centroid.

**Parameters**  $centroid$  : numpy.ndarray

Numpy array with dimensions  $1 * 2$ .

$scale$  : int

Describes the scale for the distribution.

$n$  : int

Describes the number of points to make.

**Returns**  $X$  : numpy.ndarray

A numpy array with dimensions  $n * 2$ .

`ML.KMeans.kmeans.kmeans` ( $X, k, initial\_centroids=None, draw=False, output=False$ )

Runs kmeans until clusters stop moving.

**Parameters**  $X$  : numpy.ndarray

A numpy array with 2 columns.

$k$  : int

Describes the number of centroids.

$initial\_centroids$  : numpy.ndarray, optional

A numpy array with initial centroids to run the algorithm. This array has with dimensions  $k * 2$ . If not provided, algorithm is initialized with random centroids from the data  $X$ .

$draw$  : boolean, optional

Describes whether the data is to be plotted (data must have 2 or less dimensions). The default is False.

**output** : boolean, optional

Describes whether debug info is to be printed (the default is False). Info includes current number of iterations and number of changed points over time.

**Returns** **centroids** : numpy.ndarray

Numpy array with learned centroids (dimensions are  $k * 2$ ).

**index** : numpy.ndarray

Numpy array with dimensions  $n * 1$ , where  $n$  is the number of rows in  $X$ . Each value describes the closest centroid to each data point in  $X$ .

**num\_iters** : int

Describes the number of iterations taken to run kmeans.

`ML.KMeans.kmeans.plotKMeans(X, centroids, previous, index)`

Plots the data and centroids.

This function plots the data with the current centroids and shows the movement of the centroids.

**Parameters** **X** : numpy.ndarray

A numpy array with 2 columns.

**centroids** : numpy.ndarray

A numpy array with 2 columns.

**previous** : numpy.ndarray

A numpy array with 2 columns and the same number of rows as *centroids*.

**index** : numpy.ndarray

A numpy array with 1 column.

## Module contents

### 2.1.3 ML.LinearRegression package

#### Submodules

#### ML.LinearRegression.LinearRegressionStream module

```
class ML.LinearRegression.LinearRegressionStream.LinearRegressionStream(draw,
                                                                    out-
                                                                    put,
                                                                    incre-
                                                                    men-
                                                                    tal=True,
                                                                    al-
                                                                    pha=0.01,
                                                                    fig-
                                                                    size=(15,
                                                                    8))
```

Helper class for linear regression.

This class provides train and predict functions for using linear regression with *Stream\_Learn*.

**Parameters** **draw** : boolean

Describes whether the data is to be plotted (data must have 1 dimension).

**output** : boolean

Describes whether debug info is to be printed. Info includes average error and current error.

**incremental** : boolean, optional

Describes whether the linear regression algorithm is run incrementally or not (the default is True). If incremental, then the algorithm uses incremental calculations for matrix inversion and matrix multiplication if the data has 1 feature, or stochastic gradient descent if the data has more than 1 feature. Otherwise, the algorithm uses linear algebra.

**alpha** : float, optional

Learning rate for stochastic gradient descent (the default is 0.01). Ignored if incremental is False or if incremental is True and data has 1 feature.

**figsize** : tuple, optional

A tuple containing the width and height of the plot for the map (the default is (15, 8)).

## Attributes

train	(function) The train function with signature as required by <i>Stream_Learn</i> .
predict	(function) The predict function with signature as required by <i>Stream_Learn</i> .
w	(tuple) The learned weight vector.
avg_error	(float) The average error per window of data trained.

## Methods

---

*reset()* Resets the KMeans functions and average values.

---

**reset ()**

Resets the KMeans functions and average values.

Resets: train, predict, avg\_error

## ML.LinearRegression.linear\_regression module

`ML.LinearRegression.linear_regression.evaluate_error(X, y, w)`

Returns the mean squared error.

**X** [numpy.ndarray] Numpy array of data.

**y** [numpy.ndarray] Numpy array of outputs. Dimensions are  $n * 1$ , where  $n$  is the number of rows in  $X$ .

**w** [numpy.ndarray] Numpy array with dimensions  $(m + 1) * 1$ , where  $m$  is the number of columns in  $X$ .

**Returns** float

The mean squared error



`ML.LinearRegression.linear_regression.init_plot(figsize=(15, 8))`

Initializes the plot.

**Parameters** `figsize` : tuple, optional

A tuple containing the width and height of the plot (the default is (15, 8)).

`ML.LinearRegression.linear_regression.plot(X, y, w)`

Plot X data, the actual y output, and the prediction line.

**Parameters** `X` : numpy.ndarray

Numpy array of data with 1 column.

`y` : numpy.ndarray

Numpy array of outputs. Dimensions are  $n * 1$ , where  $n$  is the number of rows in  $X$ .

`w` : numpy.ndarray

Numpy array with dimensions  $2 * 1$ .

`ML.LinearRegression.linear_regression.predict(X, w)`

Returns the prediction for one data point.

**Parameters** `X` : numpy.ndarray

Numpy array of data

`w` : numpy.ndarray

Numpy array with dimensions  $(m + 1) * 1$ , where  $m$  is the number of columns in  $X$ .

**Returns** float

The mean squared error

`ML.LinearRegression.linear_regression.train(X, y)`

Trains a linear regression model using linear algebra.

**Parameters** `X` : numpy.ndarray

Numpy array of data

`y` : numpy.ndarray

Numpy array of outputs. Dimensions are  $n * 1$ , where  $n$  is the number of rows in  $X$ .

**Returns** `w` : numpy.ndarray

Trained vector with dimensions  $(m + 1) * 1$ , where  $m$  is the number of columns in  $X$ .

`ML.LinearRegression.linear_regression.train_sgd(X, y, alpha, w=None)`

Trains a linear regression model using stochastic gradient descent.

**Parameters** `X` : numpy.ndarray

Numpy array of data

`y` : numpy.ndarray

Numpy array of outputs. Dimensions are  $n * 1$ , where  $n$  is the number of rows in  $X$ .

**alpha** : float

Describes the learning rate.

`w` : numpy.ndarray, optional

The initial  $w$  vector (the default is zero).

**Returns** `w` : `numpy.ndarray`

Trained vector with dimensions  $(m + 1) * 1$ , where  $m$  is the number of columns in  $X$ .

## Module contents

## 2.2 Submodules

## 2.3 ML.Stream\_Learn module

```
class ML.Stream_Learn.Stream_Learn(data_train, data_out, train_func, predict_func,
                                     min_window_size, max_window_size, step_size,
                                     num_features, filter_func=None, all_func=None)
```

Stream framework for machine learning.

This class supports machine learning for streaming data using PSTREAMS. Given data for training and predicting along with functions to learn and predict, this class will output a stream of predictions. Both batch and continual learning is supported.

**Parameters** `data_train` : *Stream* or `numpy.ndarray` or other

A object containing data to be trained on. In the case of *Stream*, the object contains tuples of values where each tuple represents a row of data. Each tuple must have at least `num_features` values. The object can also contain non-tuples provided `filter_func` is used to extract the tuples in correct format. In the case of a *numpy* array, the array must have at least `num_features` columns. Any additional values / columns correspond to the output  $y$  data. If this is not a *Stream* or *numpy* array, the data will not be split into  $x$  and  $y$ .

**data\_out** : *Stream*

A *Stream* object containing data to generate predictions on. The *Stream* contains tuples of values where each tuple represents a row of data and must have at least `num_features` values.

**train\_func** : function

A function that trains a model. This function takes parameters  $x$  and  $y$  data, a model object, and a `window_state` tuple, and returns a trained model object. In the case of `data_train` as a *Stream*, this function has the signature `(numpy.ndarray numpy.ndarray Object) -> (Object)`. The first parameter  $x$  will have dimensions  $i \times \text{num\_features}$ , where `min_window_size`  $\leq i \leq \text{max\_window\_size}$ . The second parameter  $y$  will have dimensions  $i \times \text{num\_outputs}$ , where `num_outputs` refers to the number of  $y$  outputs for an input. For example, `num_outputs` is 1 for 1 scalar output. For unsupervised learning, `num_outputs` is 0. In the case of `data_train` as a *numpy* array, this function has the signature `(numpy.ndarray numpy.ndarray Object) -> (Object)`. The first parameter  $x$  will have dimensions  $N \times \text{num\_features}$ , where  $N$  refers to the total number of training examples. The second parameter  $y$  will have dimensions  $N \times \text{num\_outputs}$  where `num_outputs` is defined as before. If `data_train` is none of the above, the function has the signature `(Object None Object) -> (Object)`. The first parameter is `data_train`. The third parameter is a model defined by this function. The fourth parameter is a `window_state` tuple with the values `(current_window_size, steady_state, reset, step_size, max_window_size)`, where `current_window_size` describes the number of points in the window, `steady_state` is a boolean that describes whether the window has reached `max_window_size`, and `reset` is a boolean that can be set to `True` to reset the window.

**predict\_func** : function

A function that takes as input 2 tuples corresponding to 1 row of data and a model and returns the prediction output. This function has the signature (tuple tuple Object) -> (Object). The first tuple x has *num\_features* values and the second tuple y has *num\_outputs* values, where *num\_outputs* refers to the number of y outputs for an input. In the case of unsupervised learning, y is empty.

**min\_window\_size** : int

An int specifying the minimum size of the window to train on for continual learning. This will be ignored for batch learning.

**max\_window\_size** : int

An int specifying the maximum size of the window to train on for continual learning. This will be ignored for batch learning.

**step\_size** : int

An int specifying the number of tuples to move the window by for continual learning. This will be ignored for batch learning.

**num\_features** : int

An int that describes the number of features in the data.

**filter\_func** : function, optional

A function that filters data for training. This function takes parameters x and y data and a model object, and returns a tuple with signature (boolean, tuple). The first value in the output describes if the data is to be trained on (True) or if it is an outlier (False). The second value is the tuple of data in correct format as described for *data\_train*. If *data\_train* is a *Stream* that contains tuples, this function has the signature (tuple tuple Object) -> (tuple). The first tuple x has *num\_features* values and the second tuple y has *num\_outputs* values, where *num\_outputs* refers to the number of y outputs for an input. The third parameter is a model defined by *train\_func*. If *data\_train* is a *Stream* that does not contain tuples, this function has the signature (Object None Object) -> (tuple), where the first parameter has the same type as the values in *data\_train*.

**all\_func** : function, optional

A function that processes the data for usage such as visualization. This function takes parameters x and y data, a model object, a state object, and a window\_state tuple and returns an updated state object. This function has the signature (np.ndarray np.ndarray Object Object tuple) -> (Object). The first numpy array x has dimensions i x *num\_features*, where *min\_window\_size* <= i <= *max\_window\_size*. The second numpy array y has dimensions i x *num\_outputs*, where *num\_outputs* refers to the number of y outputs for an input. The third parameter is the model object defined by *train\_func*. The fourth parameter is a state object defined by this function. The fifth parameter is a window\_state tuple with values as defined in description for *train\_func*.

**Methods**

<code>reset()</code>	Resets the training window to <i>min_window_size</i> .
<code>run()</code>	Runs the framework and returns a <i>Stream</i> of outputs.

**reset ()**

Resets the training window to *min\_window\_size*.

This function resets the training window to *min\_window\_size*. After resetting, the window has the last *min\_window\_size* points in the *Stream* *x\_train*. For example, if *max\_window\_size* is 100, *min\_window\_size* is 2, and the window contains points [1, 100], after resetting the window contains points [98, 99].

**Notes**

If `reset()` is called before the window has reached *max\_window\_size*, the window will continue increasing in size until it reaches *max\_window\_size*. Then, the window will reset to *min\_window\_size*.

**run ()**

Runs the framework and returns a *Stream* of outputs.

**Returns** *y\_predict* : *Stream*

A *Stream* containing outputs as returned by *predict\_func*.

## 2.4 Module contents

---

## MakeProcess module

---



---

## Operators module

---

This module has functions that convert operations on standard Python data structures to operations on streams.

The module has three collections of functions: (1) functions that convert operations on standard Python data structures to operations on streams. These functions operate on a list of input streams to generate a list of output streams. The functions deal with the following data structures:

1. lists,
2. individual elements of lists,
3. sliding windows, and
4. timed windows.

(2) functions that map the general case of multiple input streams and multiple output streams described above to the following special cases:

1. merge: an arbitrary number of input streams and a single output stream.
2. split: a single input stream and an arbitrary number of output streams.
3. op: a single input stream and a single output stream.
4. source: no input and an arbitrary number of output streams.

(e) sink: no output and an arbitrary number of input streams. These special cases simplify functions that need to be written for standard Python data structures. You can always use the multiple inputs and outputs case even if there is only one or no input or output; however, the functions for merge, split, op, source, and sink are simpler than the multiple input and output case.

(3) a function that provides a single common signature for converting operations on Python structures to operations on streams regardless of whether the function has no inputs, a single input stream, a list of input streams, or no outputs, a single output stream or a list of output streams. (12 October 2015. Mani. Changed initialization of output\_lists.)

`Operators.adjustable_window_agent` (*f*, *inputs*, *outputs*, *state*, *call\_streams*=None, *window\_size*=None, *step\_size*=None)

`Operators.adjustable_window_func` (*f*, *inputs*, *num\_outputs*, *state*, *call\_streams*, *window\_size*, *step\_size*)

`Operators.assert_is_list` (*x*)

`Operators.assert_is_list_of_lists` (*x*, *list\_size*=None)

`Operators.assert_is_list_of_streams` (*x*)

`Operators.assert_is_list_of_streams_or_None` (*x*)

`Operators.assert_is_list_or_None` (*x*)

`Operators.async_element_agent(f, inputs, outputs, state, call_streams, window_size, step_size)`

`Operators.async_element_func(f, inputs, num_outputs, state, call_streams=None, window_size=None, step_size=None)`

`Operators.awf(inputs, outputs, func, window_size, step_size, state=None, call_streams=None, **kwargs)`

`Operators.dynamic_window_agent(f, input_stream, output_stream, state, min_window_size, max_window_size, step_size)`

`Operators.dynamic_window_func(f, inputs, state, min_window_size, max_window_size, step_size)`

`Operators.ef(inputs, outputs, func, state=None, call_streams=None, **kwargs)`

`Operators.element_agent(f, inputs, outputs, state, call_streams, window_size, step_size)`

`Operators.element_func(f, inputs, num_outputs, state, call_streams, window_size, step_size)`

`Operators.h(f_type, *args)`

Calls the appropriate wrapper function given the name of the wrapper. The wrapper functions are `list_func`, `element_func`, `window_func`, ... for wrapper names 'list', 'element', 'window',...

`Operators.h_agent(f_type, *args)`

Calls the appropriate wrapper function given the name of the wrapper. The wrapper functions are `list_agent`, `element_agent`, `window_agent`, ... for wrapper names 'list', 'element', 'window',...

`Operators.lf(inputs, outputs, func, state=None, call_streams=None, **kwargs)`

`Operators.list_agent(f, inputs, outputs, state, call_streams, window_size, step_size)`

`Operators.list_func(f, inputs, num_outputs, state, call_streams, window_size, step_size)`

`Operators.list_index_for_timestamp(in_list, start_index, timestamp)`

A helper function for timed operators. The basic idea is to return the earliest index in `in_list.list[start_index:in_list.stop]` with a time field that is greater than or equal to `timestamp`. If no such index exists then return a negative number.

**Parameters** `in_list`: `InList`

`InList` = `namedtuple('InList', ['list', 'start', 'stop'])` A slice into a stream.

**start\_index**: nonnegative integer

A pointer into `in_list.list`

**timestamp**: number

**Returns** Returns positive integer `i` where:

either: 'FOUND TIME WINDOW IN IN\_LIST'

`i >= start_index` and `i <= in_list.stop` and `(in_list.list[start_index] >= timestamp` or  
`in_list.list[i-2][0] < timestamp <= in_list.list[i-1][0]` )  
)

or: 'NO TIME WINDOW IN IN\_LIST'

**`i < 0` (negative `i` indicates no time window) and**

**`(in_list.list[in_list.stop-1] <= timestamp` or**

**the list is empty, i.e. `(in_list.start = in_list.stop)`**

`Operators.main()`

`Operators.many_outputs_source(f_type, f, num_outputs, state, call_streams, window_size, step_size)`



`Operators.many_to_many(f_type, f, in_streams, num_outputs, state, call_streams, window_size, step_size)`

`Operators.many_to_many_agent(f_type, f, in_streams, out_streams, state, call_streams, window_size, step_size)`

`Operators.merge(f_type, f, in_streams, state, call_streams, window_size, step_size)`

`Operators.merge_agent(f_type, f, in_streams, out_stream, state, call_streams, window_size, step_size)`

`Operators.op(f_type, f, in_stream, state, call_streams, window_size, step_size)`

`Operators.op_agent(f_type, f, in_stream, out_stream, state, call_streams, window_size, step_size)`

`Operators.remove_no_value_and_open_multivalue(l)`

This function returns a list which is the same as the input parameter `l` except that (1) `_no_value` elements in `l` are deleted and (2) each `_multivalue` element in `l` is opened

i.e., for an object `_multivalue(list_x)` each element of `list_x` appears in the returned list.

`Operators.single_output_source(f_type, f, num_outputs, state, call_streams, window_size, step_size)`

`Operators.single_output_source_agent(f_type, f, out_stream, state, call_streams, window_size, step_size)`

`Operators.sink(f_type, f, in_stream, state, call_streams, window_size, step_size)`

`Operators.sink_merge(f_type, f, in_streams, state, call_streams, window_size, step_size)`

`Operators.split(f_type, f, in_stream, num_outputs, state, call_streams, window_size, step_size)`

`Operators.split_agent(f_type, f, in_stream, out_streams, state, call_streams, window_size, step_size)`

`Operators.stream_agent(inputs, outputs, f_type, f, state=None, call_streams=None, window_size=None, step_size=None)`

Provides a common signature for converting functions `f` on standard Python data structures to streams.

**Parameters** `f_type` : { 'element', 'list', 'window', 'timed', 'asynch\_element' }

`f_type` identifies the type of function `f` where `f` is the next parameter.

`f` : function

`inputs` : {Stream, list of Streams}

**When stream\_func has:** no input streams, `inputs` is None a single input Stream, `inputs` is a single Stream multiple input Streams, `inputs` is a list of Streams.

`outputs` : list of Streams

`state` : object

`state` is None or is an arbitrary object. The state captures all the information necessary to continue processing the input streams.

`call_streams` : None or list of Stream

If `call_streams` is None then the program sets it to `inputs` (converting `inputs` to a list of Stream if necessary). This function is called when, and only when any stream in `call_streams` is modified.

`window_size` : None or int

`window_size` must be a positive integer if `f_type` is 'window' or 'timed'. `window_size` is the size of the moving window on which the function operates.

`step_size` : None or int

`step_size` must be a positive integer if `f_type` is 'window' or 'timed'. `step_size` is the number of steps by which the moving window moves on each execution of the function.

**Returns** None

`Operators.stream_func` (*inputs, f\_type, f, num\_outputs, state=None, call\_streams=None, window\_size=None, step\_size=None*)

Provides a common signature for converting functions `f` on standard Python data structures to streams.

**Parameters** `f_type` : {'element', 'list', 'window', 'timed', 'asynch\_element'}

`f_type` identifies the type of function `f` where `f` is the next parameter.

`f` : function

`inputs` : {Stream, list of Streams}

**When `stream_func` has:** no input streams, `inputs` is None a single input Stream, `inputs` is a single Stream multiple input Streams, `inputs` is a list of Streams.

`num_outputs` : int

A nonnegative integer which is the number of output streams of this function.

`state` : object

`state` is None or is an arbitrary object. The state captures all the information necessary to continue processing the input streams.

`call_streams` : None or list of Stream

If `call_streams` is None then the program sets it to `inputs` (converting `inputs` to a list of Stream if necessary). This function is called when, and only when any stream in `call_streams` is modified.

`window_size` : None or int

`window_size` must be a positive integer if `f_type` is 'window' or 'timed'. `window_size` is the size of the moving window on which the function operates.

`step_size` : None or int

`step_size` must be a positive integer if `f_type` is 'window' or 'timed'. `step_size` is the number of steps by which the moving window moves on each execution of the function.

**Returns** list of Streams

Function `f` is applied to the appropriate data structure in the input streams to put values in the output streams. `stream_func` returns the output streams.

`Operators.tf` (*inputs, outputs, func, window\_size, step\_size, state=None, call\_streams=None, \*\*kwargs*)

`Operators.timed_agent` (*f, inputs, outputs, state, call\_streams, window\_size, step\_size*)

`Operators.timed_func` (*f, inputs, num\_outputs, state, call\_streams, window\_size, step\_size*)

`Operators.wf` (*inputs, outputs, func, window\_size, step\_size, state=None, call\_streams=None, \*\*kwargs*)

`Operators.window_agent` (*f, inputs, outputs, state, call\_streams, window\_size, step\_size*)

`Operators.window_func` (*f, inputs, num\_outputs, state, call\_streams, window\_size, step\_size*)

---

## RemoteQueue module

---



---

## Stream module

---

This module contains the Stream class. The Stream and Agent classes are the building blocks of PythonStreams. (12 October 2015. Mani. Fixed bug. Made `_no_value` and `_close` classes rather than object.)

```
class Stream.Stream(name='NoName',          proc_name='UnknownProcess',          initial_value=[],
                    stream_size=4096, buffer_size=1024)
```

Bases: object

A stream is a sequence of values. Agents can: (1) Append values to the tail of stream and close a stream. (2) Read a stream. (3) Subscribe to be notified when a value is added to a stream. (See Agent.py for details of agents.)

The ONLY way in which a stream can be modified is that values can be appended to its tail. The length of a stream (number of elements in its sequence) can stay the same or increase, but never decreases. If at some point, the length of a stream is *k*, then from that point onwards, the first *k* elements of the stream remain unchanged.

A stream is written by only one agent. Any number of agents can read a stream, and any number of agents can subscribe to a stream. An agent can be a reader and a subscriber and a writer of the same stream. An agent may subscribe to a stream without reading the stream's values; for example the agent may subscribe to a clock stream and the agent executes a state transition when the the clock stream has a new value, regardless of the value.

**Parameters name** : str, optional

name of the stream. Though the name is optional a named stream helps with debugging.  
default : 'NoName'

**proc\_name** : str, optional

The name of the process in which this agent executes. default: 'UnknownProcess'

**initial\_value** : list or array, optional

The list (or array) of initial values in the stream. default : []

**stream\_size: int, optional**

stream\_size must be a positive integer. It is the largest number of the most recent elements in the stream that are in main memory. default : DEFAULT\_STREAM\_SIZE

where DEFAULT\_STREAM\_SIZE is specified in SystemParameters.py

**buffer\_size** : int, optional

buffer\_size must be a positive integer. An exception may be thrown if an agent reads an element with index *i* in the stream where *i* is less than the length of the stream - buffer\_size. default : DEFAULT\_BUFFER\_SIZE\_FOR\_STREAM

specified in SystemParameters.py

## Notes

### 1.AGENTS SUBSCRIBING TO A STREAM

An agent is a state-transition automaton and the only action that an agent executes is a state transition. If agent  $x$  is a subscriber to a stream  $s$  then  $x.next()$  — a state transition of  $x$  — is invoked whenever messages are appended to  $s$ .

The only point at which an agent executes a state transition is when a stream to which the agent subscribes is modified.

**An agent  $x$  subscribes to a stream  $s$  by executing `s.call( $x$ )`.**

An agent  $x$  unsubscribes from a stream  $s$  by executing:

```
s.delete_caller(x)
```

### 2.AGENTS READING A STREAM

#### 2.1 Agent registers for reading

An agent can read a stream only after it registers with the stream as a reader. An agent  $r$  registers with a stream  $s$  by executing:

```
s.reader(r)
```

An agent  $r$  deletes its registration for reading  $s$  by executing:

```
s.delete_reader(r)
```

#### 2.2 Slice of a stream that can be read by an agent

At any given point, an agent  $r$  that has registered to read a stream  $s$  can only read some of the most recent values in the stream. The number of values that an agent can read may vary from agent to agent. A reader  $r$  can only read a slice:

```
s[s.start[r]+s.offset: s.stop+s.offset]
```

of stream  $s$  where `start[r]`, `stop` and `offset` are defined later.

### 3.WRITING A STREAM

#### 3.1 Extending a stream

When an agent is created it is passed a list of streams that it can write.

An agent adds a single element  $v$  to a stream  $s$  by executing:

```
s.append(v)
```

An agent adds the sequence of values in a list  $l$  to a stream  $s$  by executing:

```
s.extend(l)
```

The operations `append` and `extend` of streams are analogous to operations with the same names on lists.

#### 3.2 Closing a Stream

A stream is either closed or open. Initially a stream is open. An agent that writes a stream  $s$  can close  $s$  by executing:

```
s.close()
```

A closed stream cannot be modified.

### 4.MEMORY

#### 4.1 The most recent values of a stream

The most recent elements of a stream are stored in main memory. In addition, the user can specify whether all or part of the stream is saved to a file.

Associated with each stream  $s$  is a list (or array)  $s.recent$  that includes the most recent elements of  $s$ . If the value of  $s$  is a sequence:

$$s[0], \dots, s[n-1],$$

at a point in a computation then at that point,  $s.recent$  is a list

$$s[m], \dots, s[n-1]$$

for some  $m$ , followed by some padding (usually a sequence of zeroes, as described later).

The system ensures that all readers of stream  $s$  only read elements of  $s$  that are in  $s.recent$ .

#### 4.2 Slice of a stream that can be read

Associated with a reader  $r$  of stream  $s$  is an integer  $s.start[r]$ . Reader  $r$  can only read the slice:

$$s.recent[s.start[r] : ]$$

of  $s.recent$ .

For readers  $r1$  and  $r2$  of a stream  $s$  the values  $s.start[r1]$  and  $s.start[r2]$  may be different.

#### 4.3 When a reader finishes reading part of a stream

Reader  $r$  informs stream  $s$  that it will only read values with indexes greater than or equal to  $j$  in the list,  $recent$ , by executing

$$s.set\_start(r, j)$$

which causes  $s.start[r]$  to be set to  $j$ .

### 5. OPERATION

#### 5.1 Memory structure

Associated with a stream is: (1) a list,  $recent$ . (2) a nonnegative integer  $stop$  where:

1.  $recent[ : stop]$  contains the most recent values of the stream,
2. the slice  $recent[stop:]$  is padded with padding values (either 0 or 0.0).

#### 3. a nonnegative integer $s.offset$ where

$$recent[i] = stream[i + offset] \text{ for } 0 \leq i < s.stop$$

Example: if the sequence of values in a stream is:

$$0, 1, \dots, 949$$

**and  $s.offset$  is 900, then**  $s.recent[i] = s[900+i]$  for  $i$  in  $0, 1, \dots, 49$ .

**Invariant:**  $len(s) = s.offset + s.stop$

where  $len(s)$  is the number of values in stream  $s$ .

The size of  $s.recent$  is the parameter  $stream\_size$  of  $s$ . Recommendations for the value of  $stream\_size$  are given after a few paragraphs.

The maximum size of the list that an agent can read is the parameter,  $buffer\_size$ . Set  $buffer\_size$  large enough so that the size of the slice that any agent wants to read is less than  $buffer\_size$ . If an agent is slow compared to the rate at which the stream grows then the  $buffer\_size$  should be large. For example, if an agent is reading the

element in the stream at location  $i$ , and the stream has grown to  $l$  elements then `buffer_size` must be greater than  $l - i$ .

(In later implementations, if an agent reads a part of stream  $s$  that is not in `s.recent`, then the value read is obtained from values saved to a file.)

The entire stream, or the stream up to `offset`, can be saved in a file for later processing. You can also specify that no part of the stream is saved to a file. (Note, if the stream  $s$  is not saved, and any agent reads an element of the stream  $s$  that is not in main memory, then an exception is raised.)

In the current implementation old values of the stream are not saved.

## 5.2 Memory Management

We illustrate memory management with the following example with `stream_size=4` and `buffer_size=1`

Assume that a point in time, for a stream  $s$ , the list of values in the stream is `[1, 2, 3, 10, 20]`; `stream_size=4`; `s.offset=3`; `s.stop=2`; and `s.recent = [10, 20, 0, 0]`. The size `s.recent` is `stream_size` (i.e. 4). The `s.stop` (i.e. 2) most recent values in the stream are 10 followed by a later value, 20. `s[3] == 10 == s.recent[0]` `s[4] == 20 == s.recent[1]` The values in `s.recent[s.stop:]` are padded values (zeroes).

**A reader  $r$  of stream  $s$  has access to the list:** `s.recent[s.start[r] : s.stop]`

So, if for a reader  $r$ , `s.start[r]` is 0, then  $r$  has access to the two most recent values in the stream, i.e., the list `[10, 20]`. If for another reader  $q$ , `s.start[q]=1`, then  $q$  has access to the list `[20]`. And for another reader  $u$ , `s.start[q]=2`, then  $u$  has access to the empty list `[]`.

When a value  $v$  is appended to stream  $s$ , then  $v$  is inserted in `s.recent`, replacing a padded value, and `s.stop` is incremented. If the empty space (i.e., the number of padded values) in `s.recent` decreases below `buffer_size` then a new version of `s.recent` is created containing only the `buffer_size` most recent values of the stream.

Example: Start with the same example as the previous example with `buffer_size = 2`

Then a new value, 30 is appended to the stream, making the list of values in  $s$ : `[1, 2, 3, 10, 20, 30]` `s.stop = 3`; `s.offset` is unchanged (i.e. 3) and `s.recent = [10, 20, 30, 0]`. Now the size of the empty space in `s.recent` is 1, which is less than `buffer_size`. So, the program sets `s.recent` to `[20, 30, 0, 0]`, keeping the `buffer_size` (i.e. 2) most recent values in `s.recent` and removing older values from main memory, and it sets `s.stop` to `buffer_size`, and increments `offset` by `s.stop - buffer_size`. Now

`s.stop = 2` `s.offset = 4`



## Attributes

<code>recent</code>	(list) A list of the most recent values of the stream. <code>recent</code> is a NumPy array if specified.
<code>stop</code>	(int) index into the list <code>recent</code> . <code>s.recent[:s.stop]</code> contains the <code>s.stop</code> most recent values of stream <code>s</code> . <code>s.recent[s.stop:]</code> contains padded values.
<code>offset: int</code>	index into the stream used to map the location of an element in the entire stream with the location of the same element in <code>s.recent</code> , which only contains the most recent elements of the stream. For a stream <code>s</code> : <code>s.recent[i] = s[s.offset + i]</code> for <code>i</code> in <code>range(s.stop)</code>
<code>start</code>	(dict of readers.) key = reader value = start index of the reader Reader <code>r</code> can read the slice: <code>s.recent[s.start[r] : s.stop]</code> in <code>s.recent</code> which is equivalent to the following slice in the entire stream: <code>s[s.start[r]+s.offset: s.stop+s.offset]</code>
<code>sub-scribers_set: set</code>	the set of subscribers for this stream. Subscribers are agents to be notified when an element is added to the stream.
<code>closed: boolean</code>	True if and only if the stream is closed. An exception is thrown if a value is appended to a closed stream.
<code>_buffer_size: int</code>	Invariant: For every reader <code>r</code> of stream <code>s</code> : <code>s.stop - s.start[r] &lt; s._buffer_size</code> A reader can only access <code>_buffer_size</code> number of consecutive, most recent, elements in the stream.
<code>_begin</code>	(int) index into the list, <code>recent[:_begin]</code> is not being accessed by any reader; therefore <code>recent[:_begin]</code> can be deleted from main memory. Invariant: for all readers <code>r</code> : <code>_begin &lt;= min(start[r])</code>

## Methods

<code>append(value)</code>	Append a single value to the end of the stream.
<code>call(agent)</code>	Register a subscriber for this stream.
<code>close()</code>	Close this stream.”
<code>delete_caller(agent)</code>	Delete a subscriber for this stream.
<code>delete_reader(reader)</code>	Delete this reader from this stream.
<code>extend(value_list)</code>	Extend the stream by the list of values, <code>value_list</code> .
<code>print_recent()</code>	
<code>reader(reader[, start_index])</code>	Register a reader.
<code>set_name(name)</code>	
<code>set_start(reader, start)</code>	The reader tells the stream that it is only accessing elements of the list <code>recent</code> with index <code>start</code> or high

### **append** (*value*)

Append a single value to the end of the stream.

### **call** (*agent*)

Register a subscriber for this stream.

### **close** ()

Close this stream.”

### **delete\_caller** (*agent*)

Delete a subscriber for this stream.

### **delete\_reader** (*reader*)

Delete this reader from this stream.

### **extend** (*value\_list*)

Extend the stream by the list of values, `value_list`.

**Parameters** `value_list: list`

**print\_recent** ()

**reader** (*reader*, *start\_index*=0)  
Register a reader.

The newly registered reader starts reading list recent from index start, i.e., reads the slice recent[start\_index:s.stop] If reader has already been registered with this stream its start value is updated to the parameter in the call.

**set\_name** (*name*)

**set\_start** (*reader*, *start*)

The reader tells the stream that it is only accessing elements of the list recent with index start or higher.

**class** Stream.**StreamArray** (*name*=None)  
Bases: *Stream.Stream*

### Methods

<code>append(value)</code>	Append a single value to the end of the stream.
<code>call(agent)</code>	Register a subscriber for this stream.
<code>close()</code>	Close this stream.”
<code>delete_caller(agent)</code>	Delete a subscriber for this stream.
<code>delete_reader(reader)</code>	Delete this reader from this stream.
<code>extend(a)</code>	Extend the stream by an numpy ndarray.
<code>print_recent()</code>	
<code>reader(reader[, start_index])</code>	Register a reader.
<code>set_name(name)</code>	
<code>set_start(reader, start)</code>	The reader tells the stream that it is only accessing elements of the list recent with index start or high

**extend** (*a*)

Extend the stream by an numpy ndarray.

**Parameters** *a*: np.ndarray or list

**class** Stream.**StreamSeries** (*name*=None)  
Bases: *Stream.Stream*

### Methods

<code>append(value)</code>	Append a single value to the end of the stream.
<code>call(agent)</code>	Register a subscriber for this stream.
<code>close()</code>	Close this stream.”
<code>delete_caller(agent)</code>	Delete a subscriber for this stream.
<code>delete_reader(reader)</code>	Delete this reader from this stream.
<code>extend(value_list)</code>	Extend the stream by the list of values, value_list.
<code>print_recent()</code>	
<code>reader(reader[, start_index])</code>	Register a reader.
<code>set_name(name)</code>	
<code>set_start(reader, start)</code>	The reader tells the stream that it is only accessing elements of the list recent with index start or high

**class** Stream.**StreamTimed** (*name*=None)  
Bases: *Stream.Stream*

**Methods**

<code>append(value)</code>	Append a single value to the end of the stream.
<code>call(agent)</code>	Register a subscriber for this stream.
<code>close()</code>	Close this stream.”
<code>delete_caller(agent)</code>	Delete a subscriber for this stream.
<code>delete_reader(reader)</code>	Delete this reader from this stream.
<code>extend(value_list)</code>	Extend the stream by the list of values, <code>value_list</code> .
<code>print_recent()</code>	
<code>reader(reader[, start_index])</code>	Register a reader.
<code>set_name(name)</code>	
<code>set_start(reader, start)</code>	The reader tells the stream that it is only accessing elements of the list recent with index <code>start</code> or high

**class** `Stream.TimeAndValue`

Bases: `tuple`

**Attributes**

<code>time</code>	Alias for field number 0
<code>value</code>	Alias for field number 1

**Methods**

<code>count(...)</code>	
<code>index((value, [start, ...])</code>	Raises <code>ValueError</code> if the value is not present.

**time**

Alias for field number 0

**value**

Alias for field number 1



---

## SystemParameters module

---

SYSTEM\_PARAMETERS



---

## examples\_element\_wrapper module

---

This module contains examples of the ‘element’ wrapper. A function that takes a single element of an input stream and generates a single element of an output stream is an example of a function that is wrapped by the ‘element’ wrapper to create a function on streams.

The module has the following parts: (1) op or simple operator:

- single input, single output
- 2. **sink**: single input, no outputs
- 3. **source**: no input, multiple outputs
- 4. **split**: single input, multiple output
- 5. **merge**: multiple input, single output
- 6. **general case**: multiple input, multiple output

All of the first 5 cases are specializations of the general case; however, the syntactic sugar they provide can be helpful.

For each of the above cases we first consider agents that are stateless and then consider agents with state.

`examples_element_wrapper.average(v, state)`

**Parameters** **v** : number

The next element of the input stream of the agent.

**state**: (n, cumulative)

The state of the agent where n : number

The value of the next element in the agent’s input stream.

**cumulative** [number] The sum of the values that the agent has received on its input stream.

**Returns** (mean, state)

**mean** : floating point number

The average of the values received so far by the agent

**state** : (n, cumulative)

The new state of the agent.

`examples_element_wrapper.average_stream(stream)`

```
examples_element_wrapper.boolean_of_values_greater_than_threshold(stream,  
                                                                    threshold)
```

```
examples_element_wrapper.cumulative_stream(stream)
```

```
examples_element_wrapper.cumulative_sum(v, cumulative)
```

**Parameters** *v* : number

The next element of the input stream of the agent.

**cumulative:** number

The state of the agent. The state is the sum of all the values received on the agent's input stream.

**Returns** (*cumulative*, *cumulative*)

**cumulative** : number

The state after the transition, i.e., the sum of values received on the agent's input stream including the value received in this transition.

```
examples_element_wrapper.discard_odds(stream)
```

```
examples_element_wrapper.discard_odds_1(stream)
```

```
examples_element_wrapper.double(v)
```

```
examples_element_wrapper.double_stream(stream)
```

```
examples_element_wrapper.even(v)
```

```
examples_element_wrapper.even2(v)
```

```
examples_element_wrapper.even3(v)
```

```
examples_element_wrapper.even_1(v)
```

```
examples_element_wrapper.even_odd(m)
```

```
examples_element_wrapper.even_odd_stream(stream)
```

```
examples_element_wrapper.evens_and_halves(stream)
```

```
examples_element_wrapper.evens_and_halves_3(stream)
```

```
examples_element_wrapper.exp_mult_div_stream(stream, exponent, multiplier, divisor)
```

```
examples_element_wrapper.file_to_stream(filename, output_stream, time_period=0)
```

**Parameters** *filename*: str

**output\_stream**: Stream

**time\_period**: int or float, nonnegative

```
examples_element_wrapper.inrange_and_outlier_streams(x_and_y_streams,    A,    B,  
                                                                    DELTA)
```

```
examples_element_wrapper.join_streams(list_of_streams)
```

```
examples_element_wrapper.main()
```

```
examples_element_wrapper.max_and_min_seen_across_all_streams(list_of_streams)
```

```
examples_element_wrapper.max_and_min_stream(list_of_streams)
```

```
examples_element_wrapper.max_and_min_with_names(list_of_numbers, state)
```

```
examples_element_wrapper.max_of_all_inputs(list_of_elements, previous_max)
```



```
examples_element_wrapper.max_seen_across_all_streams (list_of_streams)
examples_element_wrapper.max_stream (list_of_streams)
examples_element_wrapper.max_with_index (list_of_numbers, state)
examples_element_wrapper.mean_stream (list_of_streams)
examples_element_wrapper.multiply_elements_in_stream (stream, multiplier)
examples_element_wrapper.print0 (stream)
examples_element_wrapper.print_stream (stream)
examples_element_wrapper.print_streams (list_of_streams)
examples_element_wrapper.print_sums (list_of_streams)
examples_element_wrapper.rand (output_stream, num_outputs, time_period=0)
```

**Parameters** *output\_stream*: Stream

**num\_outputs**: int, positive

**time\_period**: int or float, positive

```
examples_element_wrapper.single_stream_of_random_numbers (timer_stream)
examples_element_wrapper.square (v)
examples_element_wrapper.square_and_double (m)
examples_element_wrapper.square_and_double_stream (stream)
examples_element_wrapper.square_stream (stream)
examples_element_wrapper.stream_to_file (stream, filename)
examples_element_wrapper.temperature_and_humidity_streams (stream, DELTA)
examples_element_wrapper.timer (output_stream, num_outputs, time_period)
```

**Parameters** *stream*: Stream

**num\_outputs**: int, positive

**time\_period**: int or float, positive



---

## examples\_timed\_window\_wrapper module

---

```
examples_timed_window_wrapper.avg_of_max_and_min_in_timed_list (timed_list,
                                                                state)
examples_timed_window_wrapper.avg_of_max_and_min_values_in_all_timed_lists (list_of_timed_lists,
                                                                              state)
examples_timed_window_wrapper.diff_of_counts_in_lists (list_of_two_lists)
examples_timed_window_wrapper.exponential_smoothed_timed_windows (input_stream,
                                                                    func, alpha,
                                                                    window_size,
                                                                    step_size,
                                                                    initial_state)
```

### Parameters

**input\_stream: Stream** A previously defined stream This is the only input stream of the agent.

**func: function** func operates on a list of TimeAndValue objects and returns an object that can be smoothed exponentially.

**alpha: number** The exponential smoothing parameter.

**window\_size, step\_size, initial\_state:** Already defined.

```
examples_timed_window_wrapper.max_and_min_of_values_in_timed_list (timed_list)
examples_timed_window_wrapper.max_and_min_values_in_all_timed_lists (list_of_timed_lists)
examples_timed_window_wrapper.max_sums_timed_windows (list_of_timed_lists, state)
examples_timed_window_wrapper.sum_values_in_all_timed_lists (list_of_timed_lists)
examples_timed_window_wrapper.sum_values_in_timed_list (timed_list)
```



---

## examples\_window\_wrapper module

---

This module contains examples of the ‘window’ wrapper. A window wrapper wraps a function that has a parameter which is a list or a list of lists and that returns a value or a list of values. The wrapped function operates on a sliding window of a stream or a list of sliding windows of a list of streams, and returns a stream or a list of streams.

A sliding window is defined by a window size and a step size. An operation on a sliding window carries out its first operation only when the size of the stream is at least the window size. A window is a list of the specified size. An operation is carried out on the window; then the window is moved forward in the stream by the step size.

For example, if the operation on the window is sum, and the window size is 3 and the step size is 2, and the stream is [5, 7] at a point in time t0, then no window operation is carried at t0. If at a later time, t1, the stream is [5, 7, 8] then the sum operation is carried out on this window of size 3 to return 20 at t1. Then the window operation waits until the window steps forward by 2. If at a later time, t2, the stream is [5, 7, 8, 2], no operation is carried out at t2. At a later time t3, if the stream is [5, 7, 8, 2, 5] then an operation is carried out on the window [8, 2, 5] to give 15.

A window operation on multiple input streams waits until sliding windows are available on all the input streams. The window sizes and step sizes for all windows are identical.

The examples below deal with stateless and stateful cases of single and multiple input streams and single and multiple output streams. We don’t use windows for sources or for sinks because simple elements are adequate for this purpose. Likewise, we don’t use windows for asynchronous merges.

```
examples_window_wrapper.avg_of_difference_of_two_windows (list_of_two_windows,
                                                         state)

examples_window_wrapper.combine_windows (input_stream, weights)

examples_window_wrapper.diff_of_means_of_two_windows (list_of_two_windows)

examples_window_wrapper.exp_smoothing_mean_and_std_of_stream (stream,      alpha,
                                                             window_size,
                                                             step_size)

examples_window_wrapper.extent (lst)

examples_window_wrapper.inrange_and_outlier_streams (list_of_two_streams,    win-
                                                         dow_size,    step_size,    alpha,
                                                         threshold)

examples_window_wrapper.max_and_min (lst)

examples_window_wrapper.mean_inc (lst, state)

examples_window_wrapper.mean_of_window (stream)
```



---

## source\_stream module

---

`source_stream.main()`

`source_stream.source_stream(output_stream, number_of_values, time_period, func, **kwargs)`

Periodically appends a value to `output_stream`. The values appended are obtained by calling the function `func` and passing it keyword arguments.

If `number_of_values` is non-negative, then it is the maximum number of values inserted into `output_stream`. If `number_of_values` is negative then values are appended to `output_stream` forever.

If `time_period` is 0 then `number_of_values` must be non-negative; in this case all the values are appended to `output_stream` when `source_stream` is called.

**Parameters** `output_stream: Stream`

(Could also be a list.) The stream to which values are appended. (Note: Appending messages to a list forever will cause memory overflow.)

**number\_of\_values: int**

**time\_period: int or float, nonnegative**

The time between successive appends to `output_stream`.

**func: function**

The return value of this function is appended to `output_stream`.





---

## tutorial\_average\_of\_stream module

---

`tutorial_average_of_stream.update_avg(v, state)`



---

**tutorial\_element\_wrapper\_stream\_operator module**

---



---

## tutorial\_exponential\_smoothing module

---

`tutorial_exponential_smoothing.update_smooth` (*lst*, *state*, *h*, *alpha*)



---

**tutorial\_filter\_1 module**

---

`tutorial_filter_1.ft` (*v*, *threshold*)





---

**tutorial\_many\_to\_many module**

---



---

**tutorial\_merge module**

---

`tutorial_merge.dif` (*lst*)



---

**tutorial\_sine module**

---



---

**tutorial\_split module**

---

`tutorial_split.execute(v, g, h)`





---

**weave\_streams module**

---



---

## Indices and tables

---

- `genindex`
- `modindex`
- `search`



## a

Agent, 3

## e

examples\_element\_wrapper, 35

examples\_timed\_window\_wrapper, 39

examples\_window\_wrapper, 41

## m

ML, 16

ML.KMeans, 11

ML.KMeans.kmeans, 8

ML.KMeans.KMeansStream, 7

ML.LinearRegression, 14

ML.LinearRegression.linear\_regression,  
12

ML.LinearRegression.LinearRegressionStream,  
11

ML.Stream\_Learn, 14

## o

Operators, 19

## s

source\_stream, 43

Stream, 25

SystemParameters, 33

## t

tutorial\_average\_of\_stream, 45

tutorial\_element\_wrapper\_stream\_operator,  
47

tutorial\_exponential\_smoothing, 49

tutorial\_filter\_1, 51

tutorial\_many\_to\_many, 53

tutorial\_merge, 55

tutorial\_sine, 57

tutorial\_split, 59

## w

weave\_streams, 61



## A

`adjustable_window_agent()` (in module `Operators`), 19  
`adjustable_window_func()` (in module `Operators`), 19  
`Agent` (class in `Agent`), 3  
`Agent` (module), 3  
`append()` (`Stream.Stream` method), 29  
`assert_is_list()` (in module `Operators`), 19  
`assert_is_list_of_lists()` (in module `Operators`), 19  
`assert_is_list_of_streams()` (in module `Operators`), 19  
`assert_is_list_of_streams_or_None()` (in module `Operators`), 19  
`assert_is_list_or_None()` (in module `Operators`), 19  
`asynch_element_agent()` (in module `Operators`), 19  
`asynch_element_func()` (in module `Operators`), 20  
`average()` (in module `examples_element_wrapper`), 35  
`average_stream()` (in module `examples_element_wrapper`), 35  
`avg_of_difference_of_two_windows()` (in module `examples_window_wrapper`), 41  
`avg_of_max_and_min_in_timed_list()` (in module `examples_timed_window_wrapper`), 39  
`avg_of_max_and_min_values_in_all_timed_lists()` (in module `examples_timed_window_wrapper`), 39  
`awf()` (in module `Operators`), 20

## B

`boolean_of_values_greater_than_threshold()` (in module `examples_element_wrapper`), 35

## C

`call()` (`Stream.Stream` method), 29  
`close()` (`Stream.Stream` method), 29  
`combine_windows()` (in module `examples_window_wrapper`), 41  
`computeCentroids()` (in module `ML.KMeans.kmeans`), 8  
`cumulative_stream()` (in module `examples_element_wrapper`), 36  
`cumulative_sum()` (in module `examples_element_wrapper`), 36

## D

`delete_caller()` (`Stream.Stream` method), 29  
`delete_reader()` (`Stream.Stream` method), 29  
`dif()` (in module `tutorial_merge`), 55  
`diff_of_counts_in_lists()` (in module `examples_timed_window_wrapper`), 39  
`diff_of_means_of_two_windows()` (in module `examples_window_wrapper`), 41  
`discard_odds()` (in module `examples_element_wrapper`), 36  
`discard_odds_1()` (in module `examples_element_wrapper`), 36  
`double()` (in module `examples_element_wrapper`), 36  
`double_stream()` (in module `examples_element_wrapper`), 36  
`dynamic_window_agent()` (in module `Operators`), 20  
`dynamic_window_func()` (in module `Operators`), 20

## E

`ef()` (in module `Operators`), 20  
`element_agent()` (in module `Operators`), 20  
`element_func()` (in module `Operators`), 20  
`EPSILON` (in module `Agent`), 4  
`evaluate_error()` (in module `ML.KMeans.kmeans`), 8  
`evaluate_error()` (in module `ML.LinearRegression.linear_regression`), 12  
`even()` (in module `examples_element_wrapper`), 36  
`even2()` (in module `examples_element_wrapper`), 36  
`even3()` (in module `examples_element_wrapper`), 36  
`even_1()` (in module `examples_element_wrapper`), 36  
`even_odd()` (in module `examples_element_wrapper`), 36  
`even_odd_stream()` (in module `examples_element_wrapper`), 36  
`evens_and_halves()` (in module `examples_element_wrapper`), 36  
`evens_and_halves_3()` (in module `examples_element_wrapper`), 36  
`examples_element_wrapper` (module), 35  
`examples_timed_window_wrapper` (module), 39

examples\_window\_wrapper (module), 41  
 execute() (in module tutorial\_split), 59  
 exp\_mult\_div\_stream() (in module examples\_element\_wrapper), 36  
 exp\_smoothing\_mean\_and\_std\_of\_stream() (in module examples\_window\_wrapper), 41  
 exponential\_smoothed\_timed\_windows() (in module examples\_timed\_window\_wrapper), 39  
 extend() (Stream.Stream method), 29  
 extend() (Stream.StreamArray method), 30  
 extent() (in module examples\_window\_wrapper), 41

## F

file\_to\_stream() (in module examples\_element\_wrapper), 36  
 findClosestCentroids() (in module ML.KMeans.kmeans), 9  
 ft() (in module tutorial\_filter\_1), 51

## H

h() (in module Operators), 20  
 h\_agent() (in module Operators), 20

## I

init\_plot() (in module ML.KMeans.kmeans), 9  
 init\_plot() (in module ML.LinearRegression.linear\_regression), 12  
 initialize() (in module ML.KMeans.kmeans), 9  
 initializeCentroids() (in module ML.KMeans.kmeans), 9  
 initializeData() (in module ML.KMeans.kmeans), 10  
 initializeDataCenter() (in module ML.KMeans.kmeans), 10  
 InList (class in Agent), 4  
 inrange\_and\_outlier\_streams() (in module examples\_element\_wrapper), 36  
 inrange\_and\_outlier\_streams() (in module examples\_window\_wrapper), 41

## J

join\_streams() (in module examples\_element\_wrapper), 36

## K

kmeans() (in module ML.KMeans.kmeans), 10  
 KMeansStream (class in ML.KMeans.KMeansStream), 7

## L

lf() (in module Operators), 20  
 LinearRegressionStream (class in ML.LinearRegression.LinearRegressionStream), 11  
 list (Agent.InList attribute), 5

list\_agent() (in module Operators), 20  
 list\_func() (in module Operators), 20  
 list\_index\_for\_timestamp() (in module Operators), 20

## M

main() (in module examples\_element\_wrapper), 36  
 main() (in module Operators), 20  
 main() (in module source\_stream), 43  
 many\_outputs\_source() (in module Operators), 20  
 many\_to\_many() (in module Operators), 20  
 many\_to\_many\_agent() (in module Operators), 21  
 max\_and\_min() (in module examples\_window\_wrapper), 41  
 max\_and\_min\_of\_values\_in\_timed\_list() (in module examples\_timed\_window\_wrapper), 39  
 max\_and\_min\_seen\_across\_all\_streams() (in module examples\_element\_wrapper), 36  
 max\_and\_min\_stream() (in module examples\_element\_wrapper), 36  
 max\_and\_min\_values\_in\_all\_timed\_lists() (in module examples\_timed\_window\_wrapper), 39  
 max\_and\_min\_with\_names() (in module examples\_element\_wrapper), 36  
 max\_of\_all\_inputs() (in module examples\_element\_wrapper), 36  
 max\_seen\_across\_all\_streams() (in module examples\_element\_wrapper), 36  
 max\_stream() (in module examples\_element\_wrapper), 37  
 max\_sums\_timed\_windows() (in module examples\_timed\_window\_wrapper), 39  
 max\_with\_index() (in module examples\_element\_wrapper), 37  
 mean\_inc() (in module examples\_window\_wrapper), 41  
 mean\_of\_window() (in module examples\_window\_wrapper), 41  
 mean\_stream() (in module examples\_element\_wrapper), 37  
 merge() (in module Operators), 21  
 merge\_agent() (in module Operators), 21  
 ML (module), 16  
 ML.KMeans (module), 11  
 ML.KMeans.kmeans (module), 8  
 ML.KMeans.KMeansStream (module), 7  
 ML.LinearRegression (module), 14  
 ML.LinearRegression.linear\_regression (module), 12  
 ML.LinearRegression.LinearRegressionStream (module), 11  
 ML.Stream\_Learn (module), 14  
 multiply\_elements\_in\_stream() (in module examples\_element\_wrapper), 37

## N

next() (Agent.Agent method), 4



## O

op() (in module Operators), 21  
 op\_agent() (in module Operators), 21  
 Operators (module), 19

## P

plot() (in module ML.LinearRegression.linear\_regression), 13  
 plotKMeans() (in module ML.KMeans.kmeans), 11  
 predict() (in module ML.LinearRegression.linear\_regression), 13  
 print0() (in module examples\_element\_wrapper), 37  
 print\_recent() (Stream.Stream method), 29  
 print\_stream() (in module examples\_element\_wrapper), 37  
 print\_streams() (in module examples\_element\_wrapper), 37  
 print\_sums() (in module examples\_element\_wrapper), 37

## R

rand() (in module examples\_element\_wrapper), 37  
 reader() (Stream.Stream method), 30  
 remove\_novalue\_and\_open\_multivalue() (in module Operators), 21  
 reset() (ML.KMeans.KMeansStream.KMeansStream method), 8  
 reset() (ML.LinearRegression.LinearRegressionStream.LinearRegressionStream method), 12  
 reset() (ML.Stream\_Learn.Stream\_Learn method), 16  
 run() (ML.Stream\_Learn.Stream\_Learn method), 16

## S

set\_name() (Stream.Stream method), 30  
 set\_start() (Stream.Stream method), 30  
 single\_output\_source() (in module Operators), 21  
 single\_output\_source\_agent() (in module Operators), 21  
 single\_stream\_of\_random\_numbers() (in module examples\_element\_wrapper), 37  
 sink() (in module Operators), 21  
 sink\_merge() (in module Operators), 21  
 source\_stream (module), 43  
 source\_stream() (in module source\_stream), 43  
 split() (in module Operators), 21  
 split\_agent() (in module Operators), 21  
 square() (in module examples\_element\_wrapper), 37  
 square\_and\_double() (in module examples\_element\_wrapper), 37  
 square\_and\_double\_stream() (in module examples\_element\_wrapper), 37  
 square\_stream() (in module examples\_element\_wrapper), 37  
 start (Agent.InList attribute), 5  
 stop (Agent.InList attribute), 5

Stream (class in Stream), 25  
 Stream (module), 25  
 stream\_agent() (in module Operators), 21  
 stream\_func() (in module Operators), 22  
 Stream\_Learn (class in ML.Stream\_Learn), 14  
 stream\_to\_file() (in module examples\_element\_wrapper), 37  
 StreamArray (class in Stream), 30  
 StreamSeries (class in Stream), 30  
 StreamTimed (class in Stream), 30  
 sum\_values\_in\_all\_timed\_lists() (in module examples\_timed\_window\_wrapper), 39  
 sum\_values\_in\_timed\_list() (in module examples\_timed\_window\_wrapper), 39  
 SystemParameters (module), 33

## T

temperature\_and\_humidity\_streams() (in module examples\_element\_wrapper), 37  
 tf() (in module Operators), 22  
 time (Stream.TimeAndValue attribute), 31  
 TimeAndValue (class in Stream), 31  
 timed\_agent() (in module Operators), 22  
 timed\_func() (in module Operators), 22  
 timer() (in module examples\_element\_wrapper), 37  
 train() (in module ML.LinearRegression.linear\_regression), 13  
 train\_sgd() (in module ML.LinearRegression.linear\_regression), 13  
 tutorial\_average\_of\_stream (module), 45  
 tutorial\_element\_wrapper\_stream\_operator (module), 47  
 tutorial\_exponential\_smoothing (module), 49  
 tutorial\_filter\_1 (module), 51  
 tutorial\_many\_to\_many (module), 53  
 tutorial\_merge (module), 55  
 tutorial\_sine (module), 57  
 tutorial\_split (module), 59

## U

update\_avg() (in module tutorial\_average\_of\_stream), 45  
 update\_smooth() (in module tutorial\_exponential\_smoothing), 49

## V

value (Stream.TimeAndValue attribute), 31

## W

weave\_streams (module), 61  
 wf() (in module Operators), 22  
 window\_agent() (in module Operators), 22  
 window\_func() (in module Operators), 22