
SigFeat Documentation

Release 0.2a0

Siegfried Gündert

Aug 21, 2019

Contents

1	Examples	3
2	Highlevel API	5
2.1	Source	5
2.2	Feature	7
2.3	Extractor	27
2.4	Sink	28
2.5	Preprocess	29
3	Base level API	31
3.1	Source	31
3.2	Feature	32
3.3	Result	35
3.4	Sink	36
3.5	Preprocess	36
3.6	Parameter	37
3.7	Metadata	38
4	Indices and tables	41
5	License	43
	Python Module Index	45
	Index	47

A Signal Feature Extraction framework for Python. Focused on audio signals.

Follows a clear design:

- **Source**
- **Feature**
- **Extractor**
- **Sink**

With **Parameters**, **Metadata** handling and **Preprocessing** capability. All needed information can be written to the Sink in a clear structure.

Contents:

CHAPTER 1

Examples

On top of the base API this library already implements some useful classes:

- An `ArraySource`: Source for numpy arrays with slicing functionality. And a `SoundFileSource`: Source for sound files using `pysoundfile`.
- Commonly known spectral, temporal and other Features (e.g. `RootMeanSquare` or `SpectralFlatness` etc.)
- A `DefaultDictSink`: Sink receiving results in dictionary.

2.1 Source

class `sigfeat.source.array.ArraySource` (*array, samplerate, name="", **parameters*)
Source class for iterable arrays.

Parameters

array [ndarray] Expects an iterable array with `.shape` tuple.

samplerate [int]

name [str]

blocksize [int]

overlap [int]

Attributes

metadata Returns metadata.

parameters Returns all parameters.

Methods

<code>add_metadata(self, name, value)</code>	Appends the key value pair to metadata list.
<code>extend_metadata(self, mdata)</code>	Extends the metadata with the given list of key value pairs.
<code>fetch_metadata_as_attrs(self)</code>	Sets metadata as attributes of self.
<code><i>generate</i>(self)</code>	Returns generator that yields blocks out of the array.
<code>unroll_parameters(self, parameters)</code>	This method must be called to collect all parameters and provide them as attributes.

<code>get_class_parameters</code>	
-----------------------------------	--

generate (*self*)

Returns generator that yields blocks out of the array.

class `sigfeat.source.soundfile.SoundFileSource` (*sf=None, **parameters*)

Source generating data from SoundFiles.

The parameters are for the `SoundFile.blocks()` method used for the `blocks` method of this `Source` class.

Parameters

sf [`SoundFile` instance or `str`]

blocksize [`int`]

overlap [`int`]

frames [`int`] The number of frames to yield from soundfile. If `frames < 1`, the file is read until the end.

fill_value [`scalar`] The value last block will filled up, if it is shorter tha blocksize.

dtype [{`'float64'`, `'float32'`, `'int32'`, `'int16'`}, optional] See `soundfile.SoundFile.read()`.

always_2d [`bool`] Indicates wether all blocks are at least 2d numpy arrays.

Attributes

metadata Returns metadata.

parameters Returns all parameters.

Methods

<code>add_metadata(self, name, value)</code>	Appends the key value pair to metadata list.
<code>extend_metadata(self, mdata)</code>	Extends the metadata with the given list of key value pairs.
<code>fetch_metadata_as_attrs(self)</code>	Sets metadata as attributes of self.
<code><i>generate</i>(self)</code>	Returns generator that yields blocks from the Sound-File.
<code>unroll_parameters(self, parameters)</code>	This method must be called to collect all parameters and provide them as attributes.

<code>get_class_parameters</code>	
-----------------------------------	--

generate (*self*)

Returns generator that yields blocks from the SoundFile.

2.2 Feature

class sigfeat.feature.**Index** (*name=None, requirements=None, **parameters*)

Index of source.

Attributes

fid Returns the feature identifying tuple.

hidden Returns whether the feature is hidden or not.

metadata Returns metadata.

parameters Returns all parameters.

Methods

<code>add_metadata(self, name, value)</code>	Appends the key value pair to metadata list.
<code>dependencies(self)</code>	Yields all dependencies of this feature.
<code>extend_metadata(self, mdata)</code>	Extends the metadata with the given list of key value pairs.
<code>featureset(self[, new, autoinst, err_missing])</code>	Returns an ordered dict of all features unique in name.
<code>fetch_metadata_as_attrs(self)</code>	Sets metadata as attributes of self.
<code>gen_dependencies_instances(self[, autoinst, ...])</code>	Checks deps for being instance or class and yields instances.
<code>hide(self[, b])</code>	Hide the feature.
<code>new(self)</code>	Returns new initial feature instance with same parameters.
<code>on_finished(self, source, featureset, sink)</code>	Override this method to be run after extraction.
<code>on_start(self, source, featureset, sink)</code>	Override this method if your feature needs some initialization.
<code><i>process</i>(self, data, result)</code>	Override this method returning process results.
<code>requires(self)</code>	Override this method if your feature depends on other features.
<code>unroll_parameters(self, parameters)</code>	This method must be called to collect all parameters and provide them as attributes.
<code>validate_name(self)</code>	Checks for uniqueness of feature name in all dependent features.

get_class_parameters	
-----------------------------	--

process (*self, data, result*)

Override this method returning process results.

The data from the source will be in *data* (usually *data = block, index, ...*).

The required results from other features will be in the *result* argument (dict like *Result()*).

Parameters

data [block, index] Or the data generated from your source.

result [Result] The results from other features of current data. If you provide your required features in the `requires()` method, you will be able to access the results from those features.

Returns

res [e.g. scalar, ndarray or other custom types.] This is the feature result for the current data (block).

class `sigfeat.feature.SpectralFlux` (*name=None, requirements=None, **parameters*)
Flux of AbsRfft.

$$D_m[k] = \frac{|X_m[k]|}{\max(|X_m[k]|)} - \frac{|X_{m-1}[k]|}{\max(|X_{m-1}[k]|)}$$
$$SF X_m = \frac{1}{2} \sum_k D[k] |D[k]|$$

Parameters

axis [int] Axis along the flux will be calculated, default=0.

Attributes

fid Returns the feature identifying tuple.

hidden Returns whether the feature is hidden or not.

metadata Returns metadata.

parameters Returns all parameters.

Methods

<code>add_metadata(self, name, value)</code>	Appends the key value pair to metadata list.
<code>dependencies(self)</code>	Yields all dependencies of this feature.
<code>extend_metadata(self, mdata)</code>	Extends the metadata with the given list of key value pairs.
<code>featureset(self[, new, autoinst, err_missing])</code>	Returns an ordered dict of all features unique in name.
<code>fetch_metadata_as_attrs(self)</code>	Sets metadata as attributes of self.
<code>gen_dependencies_instances(self[, autoinst, ...])</code>	Checks deps for being instance or class and yields instances.
<code>hide(self[, b])</code>	Hide the feature.
<code>new(self)</code>	Returns new initial feature instance with same parameters.
<code>on_finished(self, source, featureset, sink)</code>	Override this method to be run after extraction.
<code>on_start(self, source, featureset, sink)</code>	Override this method if your feature needs some initialization.
<code>process(self, data, featuredata)</code>	Override this method returning process results.
<code>requires(self)</code>	Override this method if your feature depends on other features.
<code>unroll_parameters(self, parameters)</code>	This method must be called to collect all parameters and provide them as attributes.
<code>validate_name(self)</code>	Checks for uniqueness of feature name in all dependent features.

<code>get_class_parameters</code>	
-----------------------------------	--

on_start (*self*, *source*, *featureset*, *sink*)

Override this method if your feature needs some initialization.

The Extractor will give you source, featureset and sink.

Parameters

source [source]

featureset [OrderedDict of features]

sink [Sink]

process (*self*, *data*, *featuredata*)

Override this method returning process results.

The data from the source will be in *data* (usually *data* = *block*, *index*, ...).

The required results from other features will be in the *result* argument (dict like *Result()*).

Parameters

data [block, index] Or the data generated from your source.

result [Result] The results from other features of current data. If you provide your required features in the *requires()* method, you will be able to access the results from those features.

Returns

res [e.g. scalar, ndarray or other custom types.] This is the feature result for the current data (block).

requires (*self*)

Override this method if your feature depends on other features.

You can return Feature classes and Feature instances you need for your feature. Yielding is also allowed.

class sigfeat.feature.**SpectralRolloff** (*name=None*, *requirements=None*, ***parameters*)

Rolloff from AbsRfft.

The spectral rolloff is the frequency where the kappa percentage of energy is below and the 1-kappa percentage of energy is above.

TODO formula

Parameters

kappa [scalar {0...1}] Default 0.95

Attributes

fid Returns the feature identifying tuple.

hidden Returns whether the feature is hidden or not.

metadata Returns metadata.

parameters Returns all parameters.

Methods

<code>add_metadata(self, name, value)</code>	Appends the key value pair to metadata list.
<code>dependencies(self)</code>	Yields all dependencies of this feature.
<code>extend_metadata(self, mdata)</code>	Extends the metadata with the given list of key value pairs.
<code>featureset(self[, new, autoinst, err_missing])</code>	Returns an ordered dict of all features unique in name.
<code>fetch_metadata_as_attrs(self)</code>	Sets metadata as attributes of self.
<code>gen_dependencies_instances(self[, autoinst, ...])</code>	Checks deps for being instance or class and yields instances.
<code>hide(self[, b])</code>	Hide the feature.
<code>new(self)</code>	Returns new initial feature instance with same parameters.
<code>on_finished(self, source, featureset, sink)</code>	Override this method to be run after extraction.
<code>on_start(self, source, featureset, sink)</code>	Override this method if your feature needs some initialization.
<code>process(self, data, result)</code>	Override this method returning process results.
<code>requires(self)</code>	Override this method if your feature depends on other features.
<code>unroll_parameters(self, parameters)</code>	This method must be called to collect all parameters and provide them as attributes.
<code>validate_name(self)</code>	Checks for uniqueness of feature name in all dependent features.

<code>get_class_parameters</code>	
-----------------------------------	--

on_start (*self, source, featureset, sink*)

Override this method if your feature needs some initialization.

The Extractor will give you source, featureset and sink.

Parameters

source [source]

featureset [OrderedDict of features]

sink [Sink]

process (*self, data, result*)

Override this method returning process results.

The data from the source will be in `data` (usually `data = block, index, ...`).

The required results from other features will be in the `result` argument (dict like *Result()*).

Parameters

data [block, index] Or the data generated from your source.

result [Result] The results from other features of current data. If you provide your required features in the `requires()` method, you will be able to access the results from those features.

Returns

res [e.g. scalar, ndarray or other custom types.] This is the feature result for the current data (block).

requires (*self*)

Override this method if your feature depends on other features.

You can return Feature classes and Feature instances you need for your feature. Yielding is also allowed.

class sigfeat.feature.**SpectralCentroid** (*name=None, requirements=None, **parameters*)
Centroid of AbsRfft.

$$SC = \frac{\sum_k f[k] |X[k]|}{\frac{1}{K} \sum_k |X[k]|}$$

Parameters

axis [int] Axis along the centroid will be calculated, default=0.

Attributes

fid Returns the feature identifying tuple.

hidden Returns whether the feature is hidden or not.

metadata Returns metadata.

parameters Returns all parameters.

Methods

<code>add_metadata(self, name, value)</code>	Appends the key value pair to metadata list.
<code>dependencies(self)</code>	Yields all dependencies of this feature.
<code>extend_metadata(self, mdata)</code>	Extends the metadata with the given list of key value pairs.
<code>featureset(self[, new, autoinst, err_missing])</code>	Returns an ordered dict of all features unique in name.
<code>fetch_metadata_as_attrs(self)</code>	Sets metadata as attributes of self.
<code>gen_dependencies_instances(self[, autoinst, ...])</code>	Checks deps for being instance or class and yields instances.
<code>hide(self[, b])</code>	Hide the feature.
<code>new(self)</code>	Returns new initial feature instance with same parameters.
<code>on_finished(self, source, featureset, sink)</code>	Override this method to be run after extraction.
<code>on_start(self, source, featureset, sink)</code>	Override this method if your feature needs some initialization.
<code>process(self, data, resd)</code>	Override this method returning process results.
<code>requires(self)</code>	Override this method if your feature depends on other features.
<code>unroll_parameters(self, parameters)</code>	This method must be called to collect all parameters and provide them as attributes.
<code>validate_name(self)</code>	Checks for uniqueness of feature name in all dependent features.

centroid	
get_class_parameters	

on_start (*self, source, featureset, sink*)

Override this method if your feature needs some initialization.

The Extractor will give you source, featureset and sink.

Parameters

source [source]

featureset [OrderedDict of features]

sink [Sink]

process (*self*, *data*, *resd*)

Override this method returning process results.

The data from the source will be in *data* (usually *data* = *block*, *index*, ...).

The required results from other features will be in the *result* argument (dict like *Result()*).

Parameters

data [block, index] Or the data generated from your source.

result [Result] The results from other features of current data. If you provide your required features in the *requires()* method, you will be able to access the results from those features.

Returns

res [e.g. scalar, ndarray or other custom types.] This is the feature result for the current data (block).

requires (*self*)

Override this method if your feature depends on other features.

You can return Feature classes and Feature instances you need for your feature. Yielding is also allowed.

class sigfeat.feature.SpectralSpread (*name=None*, *requirements=None*, ***parameters*)
Spread of AbsRfft.

$$SSP_m = \frac{\sum_k (f[k]SC_m[k])^2 |X_m[k]|}{\sum_k |X_m[k]|}$$

Parameters

axis [int] Axis along the centroid will be calculated, default=0.

Attributes

fid Returns the feature identifying tuple.

hidden Returns whether the feature is hidden or not.

metadata Returns metadata.

parameters Returns all parameters.

Methods

<code>add_metadata(self, name, value)</code>	Appends the key value pair to metadata list.
<code>dependencies(self)</code>	Yields all dependencies of this feature.
<code>extend_metadata(self, mdata)</code>	Extends the metadata with the given list of key value pairs.
<code>featureset(self[, new, autoinst, err_missing])</code>	Returns an ordered dict of all features unique in name.

Continued on next page

Table 7 – continued from previous page

<code>fetch_metadata_as_attrs(self)</code>	Sets metadata as attributes of self.
<code>gen_dependencies_instances(self[, autoinst, ...])</code>	Checks deps for being instance or class and yields instances.
<code>hide(self[, b])</code>	Hide the feature.
<code>new(self)</code>	Returns new initial feature instance with same parameters.
<code>on_finished(self, source, featureset, sink)</code>	Override this method to be run after extraction.
<code>on_start(self, source, featureset, sink)</code>	Override this method if your feature needs some initialization.
<code>process(self, data, resd)</code>	Override this method returning process results.
<code>requires(self)</code>	Override this method if your feature depends on other features.
<code>unroll_parameters(self, parameters)</code>	This method must be called to collect all parameters and provide them as attributes.
<code>validate_name(self)</code>	Checks for uniqueness of feature name in all dependent features.

centroid	
get_class_parameters	
spread	

process (*self*, *data*, *resd*)

Override this method returning process results.

The data from the source will be in *data* (usually *data* = *block*, *index*, ...).

The required results from other features will be in the *result* argument (dict like *Result()*).

Parameters

data [block, index] Or the data generated from your source.

result [Result] The results from other features of current data. If you provide your required features in the *requires()* method, you will be able to access the results from those features.

Returns

res [e.g. scalar, ndarray or other custom types.] This is the feature result for the current data (block).

requires (*self*)

Override this method if your feature depends on other features.

You can return Feature classes and Feature instances you need for your feature. Yielding is also allowed.

class `sigfeat.feature.SpectralSkewness` (*name=None*, *requirements=None*, ***parameters*)
Skewness of AbsRfft.

$$SSK_m = \frac{\sum_k (f[k] - SC_m[k])^3 |X_m[k]|}{\sqrt{SSP_m^3} \sum_k |X_m[k]|}$$

Parameters

axis [int] Axis along the centroid will be calculated, default=0.

Attributes

fid Returns the feature identifying tuple.

hidden Returns whether the feature is hidden or not.

metadata Returns metadata.

parameters Returns all parameters.

Methods

<code>add_metadata(self, name, value)</code>	Appends the key value pair to metadata list.
<code>dependencies(self)</code>	Yields all dependencies of this feature.
<code>extend_metadata(self, mdata)</code>	Extends the metadata with the given list of key value pairs.
<code>featureset(self[, new, autoinst, err_missing])</code>	Returns an ordered dict of all features unique in name.
<code>fetch_metadata_as_attrs(self)</code>	Sets metadata as attributes of self.
<code>gen_dependencies_instances(self[, autoinst, ...])</code>	Checks deps for being instance or class and yields instances.
<code>hide(self[, b])</code>	Hide the feature.
<code>new(self)</code>	Returns new initial feature instance with same parameters.
<code>on_finished(self, source, featureset, sink)</code>	Override this method to be run after extraction.
<code>on_start(self, source, featureset, sink)</code>	Override this method if your feature needs some initialization.
<code><i>process</i>(self, data, resd)</code>	Override this method returning process results.
<code><i>requires</i>(self)</code>	Override this method if your feature depends on other features.
<code>unroll_parameters(self, parameters)</code>	This method must be called to collect all parameters and provide them as attributes.
<code>validate_name(self)</code>	Checks for uniqueness of feature name in all dependent features.

centroid	
get_class_parameters	
skewness	

process (*self*, *data*, *resd*)

Override this method returning process results.

The data from the source will be in *data* (usually *data* = *block*, *index*, ...).

The required results from other features will be in the *result* argument (dict like *Result()*).

Parameters

data [*block*, *index*] Or the data generated from your source.

result [*Result*] The results from other features of current data. If you provide your required features in the *requires()* method, you will be able to access the results from those features.

Returns

res [e.g. scalar, ndarray or other custom types.] This is the feature result for the current data (*block*).

requires (*self*)

Override this method if your feature depends on other features.

You can return Feature classes and Feature instances you need for your feature. Yielding is also allowed.

class sigfeat.feature.**SpectralKurtosis** (*name=None, requirements=None, **parameters*)
Kurtosis of AbsRfft.

$$SK_m = \frac{\sum_k (f[k] - SC_m[k])^4 |X_m[k]|}{SSP_m^2 \sum_k |X_m[k]|}$$

Parameters

axis [int] Axis along the centroid will be calculated, default=0.

Attributes

fid Returns the feature identifying tuple.

hidden Returns whether the feature is hidden or not.

metadata Returns metadata.

parameters Returns all parameters.

Methods

<code>add_metadata(self, name, value)</code>	Appends the key value pair to metadata list.
<code>dependencies(self)</code>	Yields all dependencies of this feature.
<code>extend_metadata(self, mdata)</code>	Extends the metadata with the given list of key value pairs.
<code>featureset(self[, new, autoinst, err_missing])</code>	Returns an ordered dict of all features unique in name.
<code>fetch_metadata_as_attrs(self)</code>	Sets metadata as attributes of self.
<code>gen_dependencies_instances(self[, autoinst, ...])</code>	Checks deps for being instance or class and yields instances.
<code>hide(self[, b])</code>	Hide the feature.
<code>new(self)</code>	Returns new initial feature instance with same parameters.
<code>on_finished(self, source, featureset, sink)</code>	Override this method to be run after extraction.
<code>on_start(self, source, featureset, sink)</code>	Override this method if your feature needs some initialization.
<code>process(self, data, resd)</code>	Override this method returning process results.
<code>requires(self)</code>	Override this method if your feature depends on other features.
<code>unroll_parameters(self, parameters)</code>	This method must be called to collect all parameters and provide them as attributes.
<code>validate_name(self)</code>	Checks for uniqueness of feature name in all dependent features.

centroid	
get_class_parameters	
kurtosis	

process (*self, data, resd*)

Override this method returning process results.

The data from the source will be in `data` (usually `data = block, index, ...`).

The required results from other features will be in the result argument (dict like *Result()*).

Parameters

data [block, index] Or the data generated from your source.

result [Result] The results from other features of current data. If you provide your required features in the *requires()* method, you will be able to access the results from those features.

Returns

res [e.g. scalar, ndarray or other custom types.] This is the feature result for the current data (block).

requires (*self*)

Override this method if your feature depends on other features.

You can return Feature classes and Feature instances you need for your feature. Yielding is also allowed.

class sigfeat.feature.**SpectralFlatness** (*name=None, requirements=None, **parameters*)
Flatness of AbsRfft.

$$SF_m = \frac{(\prod_k |X_m[k]|)^{1/K}}{\frac{1}{K} \sum_k |X_m[k]|}$$

Parameters

axis [int] Axis along the flatness will be calculated, default=0.

Attributes

fid Returns the feature identifying tuple.

hidden Returns whether the feature is hidden or not.

metadata Returns metadata.

parameters Returns all parameters.

Methods

<code>add_metadata(self, name, value)</code>	Appends the key value pair to metadata list.
<code>dependencies(self)</code>	Yields all dependencies of this feature.
<code>extend_metadata(self, mdata)</code>	Extends the metadata with the given list of key value pairs.
<code>featureset(self[, new, autoinst, err_missing])</code>	Returns an ordered dict of all features unique in name.
<code>fetch_metadata_as_attrs(self)</code>	Sets metadata as attributes of self.
<code>gen_dependencies_instances(self[, autoinst, ...])</code>	Checks deps for being instance or class and yields instances.
<code>hide(self[, b])</code>	Hide the feature.
<code>new(self)</code>	Returns new initial feature instance with same parameters.
<code>on_finished(self, source, featureset, sink)</code>	Override this method to be run after extraction.
<code>on_start(self, source, featureset, sink)</code>	Override this method if your feature needs some initialization.
<code>process(self, data, featuredata)</code>	Override this method returning process results.
<code>requires(self)</code>	Override this method if your feature depends on other features.

Continued on next page

Table 10 – continued from previous page

<code>unroll_parameters(self, parameters)</code>	This method must be called to collect all parameters and provide them as attributes.
<code>validate_name(self)</code>	Checks for uniqueness of feature name in all dependent features.

<code>get_class_parameters</code>	
-----------------------------------	--

process (*self*, *data*, *featuredata*)

Override this method returning process results.

The data from the source will be in *data* (usually *data* = *block*, *index*, ...).

The required results from other features will be in the *result* argument (dict like *Result()*).

Parameters

data [block, index] Or the data generated from your source.

result [Result] The results from other features of current data. If you provide your required features in the *requires()* method, you will be able to access the results from those features.

Returns

res [e.g. scalar, ndarray or other custom types.] This is the feature result for the current data (block).

requires (*self*)

Override this method if your feature depends on other features.

You can return Feature classes and Feature instances you need for your feature. Yielding is also allowed.

class `sigfeat.feature.SpectralCrestFactor` (*name=None*, *requirements=None*, ***parameters*)

Crest Factor of AbsRfft

$$SCF_m = \frac{\max(|X_m|)}{X_{m,RMS}}$$

Attributes

fid Returns the feature identifying tuple.

hidden Returns whether the feature is hidden or not.

metadata Returns metadata.

parameters Returns all parameters.

Methods

<code>add_metadata(self, name, value)</code>	Appends the key value pair to metadata list.
<code>dependencies(self)</code>	Yields all dependencies of this feature.
<code>extend_metadata(self, mdata)</code>	Extends the metadata with the given list of key value pairs.
<code>featureset(self[, new, autoinst, err_missing])</code>	Returns an ordered dict of all features unique in name.
<code>fetch_metadata_as_attrs(self)</code>	Sets metadata as attributes of self.

Continued on next page

Table 11 – continued from previous page

<code>gen_dependencies_instances(self[, au- toinst, ...])</code>	Checks deps for being instance or class and yields instances.
<code>hide(self[, b])</code>	Hide the feature.
<code>new(self)</code>	Returns new initial feature instance with same parameters.
<code>on_finished(self, source, featureset, sink)</code>	Override this method to be run after extraction.
<code>on_start(self, source, featureset, sink)</code>	Override this method if your feature needs some initialization.
<code>process(self, data, result)</code>	Override this method returning process results.
<code>requires(self)</code>	Override this method if your feature depends on other features.
<code>unroll_parameters(self, parameters)</code>	This method must be called to collect all parameters and provide them as attributes.
<code>validate_name(self)</code>	Checks for uniqueness of feature name in all dependent features.

<code>get_class_parameters</code>	
-----------------------------------	--

process (*self*, *data*, *result*)

Override this method returning process results.

The data from the source will be in *data* (usually *data* = *block*, *index*, ...).

The required results from other features will be in the *result* argument (dict like *Result()*).

Parameters

data [block, index] Or the data generated from your source.

result [Result] The results from other features of current data. If you provide your required features in the *requires()* method, you will be able to access the results from those features.

Returns

res [e.g. scalar, ndarray or other custom types.] This is the feature result for the current data (block).

requires (*self*)

Override this method if your feature depends on other features.

You can return Feature classes and Feature instances you need for your feature. Yielding is also allowed.

class `sigfeat.feature.SpectralSlope` (*name=None*, *requirements=None*, ***parameters*)

Attributes

fid Returns the feature identifying tuple.

hidden Returns whether the feature is hidden or not.

metadata Returns metadata.

parameters Returns all parameters.

Methods

<code>add_metadata(self, name, value)</code>	Appends the key value pair to metadata list.
<code>dependencies(self)</code>	Yields all dependencies of this feature.
<code>extend_metadata(self, mdata)</code>	Extends the metadata with the given list of key value pairs.
<code>featureset(self[, new, autoinst, err_missing])</code>	Returns an ordered dict of all features unique in name.
<code>fetch_metadata_as_attrs(self)</code>	Sets metadata as attributes of self.
<code>gen_dependencies_instances(self[, autoinst, ...])</code>	Checks deps for being instance or class and yields instances.
<code>hide(self[, b])</code>	Hide the feature.
<code>new(self)</code>	Returns new initial feature instance with same parameters.
<code>on_finished(self, source, featureset, sink)</code>	Override this method to be run after extraction.
<code>on_start(self, source, features, sink)</code>	Override this method if your feature needs some initialization.
<code>process(self, data, resd)</code>	Override this method returning process results.
<code>requires(self)</code>	Override this method if your feature depends on other features.
<code>unroll_parameters(self, parameters)</code>	This method must be called to collect all parameters and provide them as attributes.
<code>validate_name(self)</code>	Checks for uniqueness of feature name in all dependent features.

get_class_parameters	
-----------------------------	--

on_start (*self, source, features, sink*)

Override this method if your feature needs some initialization.

The Extractor will give you source, featureset and sink.

Parameters

source [source]

featureset [OrderedDict of features]

sink [Sink]

process (*self, data, resd*)

Override this method returning process results.

The data from the source will be in *data* (usually *data = block, index, ...*).

The required results from other features will be in the *result* argument (dict like *Result()*).

Parameters

data [block, index] Or the data generated from your source.

result [Result] The results from other features of current data. If you provide your required features in the *requires()* method, you will be able to access the results from those features.

Returns

res [e.g. scalar, ndarray or other custom types.] This is the feature result for the current data (block).

requires (*self*)

Override this method if your feature depends on other features.

You can return Feature classes and Feature instances you need for your feature. Yielding is also allowed.

class sigfeat.feature.**MFCC** (*name=None, requirements=None, **parameters*)

Attributes

fid Returns the feature identifying tuple.

hidden Returns whether the feature is hidden or not.

metadata Returns metadata.

parameters Returns all parameters.

Methods

<code>add_metadata(self, name, value)</code>	Appends the key value pair to metadata list.
<code>dependencies(self)</code>	Yields all dependencies of this feature.
<code>extend_metadata(self, mdata)</code>	Extends the metadata with the given list of key value pairs.
<code>featureset(self[, new, autoinst, err_missing])</code>	Returns an ordered dict of all features unique in name.
<code>fetch_metadata_as_attrs(self)</code>	Sets metadata as attributes of self.
<code>gen_dependencies_instances(self[, autoinst, ...])</code>	Checks deps for being instance or class and yields instances.
<code>hide(self[, b])</code>	Hide the feature.
<code>new(self)</code>	Returns new initial feature instance with same parameters.
<code>on_finished(self, source, featureset, sink)</code>	Override this method to be run after extraction.
<code>on_start(self, source, featureset, sink)</code>	Override this method if your feature needs some initialization.
<code>process(self, data, resd)</code>	Override this method returning process results.
<code>requires(self)</code>	Override this method if your feature depends on other features.
<code>unroll_parameters(self, parameters)</code>	This method must be called to collect all parameters and provide them as attributes.
<code>validate_name(self)</code>	Checks for uniqueness of feature name in all dependent features.

get_class_parameters	
-----------------------------	--

process (*self, data, resd*)

Override this method returning process results.

The data from the source will be in *data* (usually *data = block, index, ...*).

The required results from other features will be in the *result* argument (dict like *Result()*).

Parameters

data [block, index] Or the data generated from your source.

result [Result] The results from other features of current data. If you provide your required features in the *requires()* method, you will be able to access the results from those features.

Returns

res [e.g. scalar, ndarray or other custom types.] This is the feature result for the current data (block).

requires (*self*)

Override this method if your feature depends on other features.

You can return Feature classes and Feature instances you need for your feature. Yielding is also allowed.

class sigfeat.feature.**RootMeanSquare** (*name=None, requirements=None, **parameters*)
Root Mean Square (RMS) from Source data.

$$RMS_m = \sqrt{\frac{1}{N} \sum_n x_m[n]^2}$$

Parameters

axis [int] Axis along which the feature is computed.

Attributes

fid Returns the feature identifying tuple.

hidden Returns whether the feature is hidden or not.

metadata Returns metadata.

parameters Returns all parameters.

Methods

<code>add_metadata(self, name, value)</code>	Appends the key value pair to metadata list.
<code>dependencies(self)</code>	Yields all dependencies of this feature.
<code>extend_metadata(self, mdata)</code>	Extends the metadata with the given list of key value pairs.
<code>featureset(self[, new, autoinst, err_missing])</code>	Returns an ordered dict of all features unique in name.
<code>fetch_metadata_as_attrs(self)</code>	Sets metadata as attributes of self.
<code>gen_dependencies_instances(self[, autoinst, ...])</code>	Checks deps for being instance or class and yields instances.
<code>hide(self[, b])</code>	Hide the feature.
<code>new(self)</code>	Returns new initial feature instance with same parameters.
<code>on_finished(self, source, featureset, sink)</code>	Override this method to be run after extraction.
<code>on_start(self, source, featureset, sink)</code>	Override this method if your feature needs some initialization.
<code>process(self, data, result)</code>	Override this method returning process results.
<code>requires(self)</code>	Override this method if your feature depends on other features.
<code>unroll_parameters(self, parameters)</code>	This method must be called to collect all parameters and provide them as attributes.
<code>validate_name(self)</code>	Checks for uniqueness of feature name in all dependent features.

<code>get_class_parameters</code>	
-----------------------------------	--

process (*self*, *data*, *result*)

Override this method returning process results.

The data from the source will be in *data* (usually *data* = *block*, *index*, ...).

The required results from other features will be in the *result* argument (dict like *Result()*).

Parameters

data [*block*, *index*] Or the data generated from your source.

result [*Result*] The results from other features of current data. If you provide your required features in the *requires()* method, you will be able to access the results from those features.

Returns

res [e.g. scalar, ndarray or other custom types.] This is the feature result for the current data (*block*).

requires (*self*)

Override this method if your feature depends on other features.

You can return Feature classes and Feature instances you need for your feature. Yielding is also allowed.

class sigfeat.feature.CrestFactor (*name=None*, *requirements=None*, ***parameters*)

Crest Factor of Source data.

Depends on Peak and RootMeanSquare features.

$$CF_m = \frac{\max(|x_m|)}{x_{m,RMS}}$$

Parameters

axis: int

Attributes

fid Returns the feature identifying tuple.

hidden Returns whether the feature is hidden or not.

metadata Returns metadata.

parameters Returns all parameters.

Methods

<code>add_metadata(self, name, value)</code>	Appends the key value pair to metadata list.
<code>dependencies(self)</code>	Yields all dependencies of this feature.
<code>extend_metadata(self, mdata)</code>	Extends the metadata with the given list of key value pairs.
<code>featureset(self[, new, autoinst, err_missing])</code>	Returns an ordered dict of all features unique in name.
<code>fetch_metadata_as_attrs(self)</code>	Sets metadata as attributes of self.
<code>gen_dependencies_instances(self[, autoinst, ...])</code>	Checks deps for being instance or class and yields instances.
<code>hide(self[, b])</code>	Hide the feature.
<code>new(self)</code>	Returns new initial feature instance with same parameters.

Continued on next page

Table 15 – continued from previous page

<code>on_finished(self, source, featureset, sink)</code>	Override this method to be run after extraction.
<code>on_start(self, source, featureset, sink)</code>	Override this method if your feature needs some initialization.
<code>process(self, data, resd)</code>	Override this method returning process results.
<code>requires(self)</code>	Override this method if your feature depends on other features.
<code>unroll_parameters(self, parameters)</code>	This method must be called to collect all parameters and provide them as attributes.
<code>validate_name(self)</code>	Checks for uniqueness of feature name in all dependent features.

<code>get_class_parameters</code>	
-----------------------------------	--

process (*self*, *data*, *resd*)

Override this method returning process results.

The data from the source will be in *data* (usually *data* = *block*, *index*, ...).

The required results from other features will be in the *result* argument (dict like *Result()*).

Parameters

data [block, index] Or the data generated from your source.

result [Result] The results from other features of current data. If you provide your required features in the *requires()* method, you will be able to access the results from those features.

Returns

res [e.g. scalar, ndarray or other custom types.] This is the feature result for the current data (block).

requires (*self*)

Override this method if your feature depends on other features.

You can return Feature classes and Feature instances you need for your feature. Yielding is also allowed.

class `sigfeat.feature.ZeroCrossingRate` (*name=None*, *requirements=None*, ***parameters*)
Zero Crossings Rate of Source data.

$$ZCR_m = \frac{f_s}{N} [0.5 + \frac{1}{2} \sum_n |\text{sgn}(x_m[n+1]) - \text{sgn}(x_m[n])|]$$

Attributes

fid Returns the feature identifying tuple.

hidden Returns whether the feature is hidden or not.

metadata Returns metadata.

parameters Returns all parameters.

Methods

<code>add_metadata(self, name, value)</code>	Appends the key value pair to metadata list.
--	--

Continued on next page

Table 16 – continued from previous page

<code>dependencies(self)</code>	Yields all dependencies of this feature.
<code>extend_metadata(self, mdata)</code>	Extends the metadata with the given list of key value pairs.
<code>featureset(self[, new, autoinst, err_missing])</code>	Returns an ordered dict of all features unique in name.
<code>fetch_metadata_as_attrs(self)</code>	Sets metadata as attributes of self.
<code>gen_dependencies_instances(self[, autoinst, ...])</code>	Checks deps for being instance or class and yields instances.
<code>hide(self[, b])</code>	Hide the feature.
<code>new(self)</code>	Returns new initial feature instance with same parameters.
<code>on_finished(self, source, featureset, sink)</code>	Override this method to be run after extraction.
<code>on_start(self, source, featureset, sink)</code>	Override this method if your feature needs some initialization.
<code>process(self, data, result)</code>	Override this method returning process results.
<code>requires(self)</code>	Override this method if your feature depends on other features.
<code>unroll_parameters(self, parameters)</code>	This method must be called to collect all parameters and provide them as attributes.
<code>validate_name(self)</code>	Checks for uniqueness of feature name in all dependent features.

get_class_parameters	
-----------------------------	--

on_start (*self, source, featureset, sink*)

Override this method if your feature needs some initialization.

The Extractor will give you source, featureset and sink.

Parameters

source [source]

featureset [OrderedDict of features]

sink [Sink]

process (*self, data, result*)

Override this method returning process results.

The data from the source will be in data (usually `data = block, index, ...`).

The required results from other features will be in the result argument (dict like *Result()*).

Parameters

data [block, index] Or the data generated from your source.

result [Result] The results from other features of current data. If you provide your required features in the `requires()` method, you will be able to access the results from those features.

Returns

res [e.g. scalar, ndarray or other custom types.] This is the feature result for the current data (block).

class sigfeat.feature.**Peak** (*name=None, requirements=None, **parameters*)
 Peak of AbsSignal from Source data.

$$P_m = \max(|x_m|)$$

Parameters

axis [int] Axis along which the feature is computed.

Attributes

fid Returns the feature identifying tuple.

hidden Returns whether the feature is hidden or not.

metadata Returns metadata.

parameters Returns all parameters.

Methods

<code>add_metadata(self, name, value)</code>	Appends the key value pair to metadata list.
<code>dependencies(self)</code>	Yields all dependencies of this feature.
<code>extend_metadata(self, mdata)</code>	Extends the metadata with the given list of key value pairs.
<code>featureset(self[, new, autoinst, err_missing])</code>	Returns an ordered dict of all features unique in name.
<code>fetch_metadata_as_attrs(self)</code>	Sets metadata as attributes of self.
<code>gen_dependencies_instances(self[, autoinst, ...])</code>	Checks deps for being instance or class and yields instances.
<code>hide(self[, b])</code>	Hide the feature.
<code>new(self)</code>	Returns new initial feature instance with same parameters.
<code>on_finished(self, source, featureset, sink)</code>	Override this method to be run after extraction.
<code>on_start(self, source, featureset, sink)</code>	Override this method if your feature needs some initialization.
<code><i>process</i>(self, data, result)</code>	Override this method returning process results.
<code><i>requires</i>(self)</code>	Override this method if your feature depends on other features.
<code>unroll_parameters(self, parameters)</code>	This method must be called to collect all parameters and provide them as attributes.
<code>validate_name(self)</code>	Checks for uniqueness of feature name in all dependent features.

get_class_parameters	
-----------------------------	--

process (*self, data, result*)

Override this method returning process results.

The data from the source will be in `data` (usually `data = block, index, ...`).

The required results from other features will be in the `result` argument (dict like *Result()*).

Parameters

data [block, index] Or the data generated from your source.

result [Result] The results from other features of current data. If you provide your required features in the `requires()` method, you will be able to access the results from those features.

Returns

res [e.g. scalar, ndarray or other custom types.] This is the feature result for the current data (block).

requires (*self*)

Override this method if your feature depends on other features.

You can return Feature classes and Feature instances you need for your feature. Yielding is also allowed.

class `sigfeat.feature.Delta` (*feature, **parameters*)

Returns a diffenetiatiated version of the given feature.

Attributes

fid Returns the feature identifying tuple.

hidden Returns whether the feature is hidden or not.

metadata Returns metadata.

parameters Returns all parameters.

Methods

<code>add_metadata(self, name, value)</code>	Appends the key value pair to metadata list.
<code>dependencies(self)</code>	Yields all dependencies of this feature.
<code>extend_metadata(self, mdata)</code>	Extends the metadata with the given list of key value pairs.
<code>featureset(self[, new, autoinst, err_missing])</code>	Returns an ordered dict of all features unique in name.
<code>fetch_metadata_as_attrs(self)</code>	Sets metadata as attributes of self.
<code>gen_dependencies_instances(self[, autoinst, ...])</code>	Checks deps for being instance or class and yields instances.
<code>hide(self[, b])</code>	Hide the feature.
<code>new(self)</code>	Returns new initial feature instance with same parameters.
<code>on_finished(self, source, featureset, sink)</code>	Override this method to be run after extraction.
<code>on_start(self, source, featureset, sink)</code>	Override this method if your feature needs some initialization.
<code>process(self, data, resultd)</code>	Override this method returning process results.
<code>requires(self)</code>	Override this method if your feature depends on other features.
<code>unroll_parameters(self, parameters)</code>	This method must be called to collect all parameters and provide them as attributes.
<code>validate_name(self)</code>	Checks for uniqueness of feature name in all dependent features.

<code>get_class_parameters</code>	
-----------------------------------	--

process (*self, data, resultd*)

Override this method returning process results.

The data from the source will be in `data` (usually `data = block, index, ...`).

The required results from other features will be in the `result` argument (dict like *Result()*).

Parameters

data [block, index] Or the data generated from your source.

result [Result] The results from other features of current data. If you provide your required features in the `requires()` method, you will be able to access the results from those features.

Returns

res [e.g. scalar, ndarray or other custom types.] This is the feature result for the current data (block).

requires (*self*)

Override this method if your feature depends on other features.

You can return Feature classes and Feature instances you need for your feature. Yielding is also allowed.

2.3 Extractor

This Module contains the central Extractor class.

The Extractor consumes features instances and is used to extract the Features from given sources into given Sink.

class `sigfeat.extractor.Extractor` (**features, autoinst=True*)

Feature Extractor

Parameters

***features** [Feature instances] Provide multiple feature instances. If you provide a Feature required by other features, please place it before the others.

autoinst [bool] Autoinstantiate required feature Classes if no instance exists in featureset. If False you will get errors with a hint which feature instance you need provide as well.

Methods

<code>extract(self, source[, sink])</code>	Extracts features from given source into given sink.
<code>get_features_parameters_and_metadata(self, source[, sink[, ...]])</code>	Returns dict with parameters and metadata from self.featureset.
<code>get_parameters_and_metadata(obj)</code>	Returns dict with parameters and metadata from given obj.
<code>reset(self)</code>	Resets the states of features.

extract (*self, source, sink=None*)

Extracts features from given source into given sink.

Parameters

source [Source instance]

sink [Sink instance]

Returns

res [Sink] The sink with processed data and metadata.

get_features_parameters_and_metadata (*self*, *hidden=False*)

Returns dict with parameters and metadata from self.featureset.

static get_parameters_and_metadata (*obj*)

Returns dict with parameters and metadata from given obj.

reset (*self*)

Resets the states of features.

If a new source shall be processed this may be usefull or needed.

2.4 Sink

class sigfeat.sink.default.DefaultDictSink

The receive_append method appends input to 'results' defaultdict(list)

Methods

clear()	
copy()	
fromkeys(iterable[, value])	Create a new dictionary with keys from iterable and values set to value.
get(self, key[, default])	Return the value for key if key is in the dictionary, else default.
items()	
keys()	
pop()	If key is not found, d is returned if given, otherwise KeyError is raised
popitem()	2-tuple; but raise KeyError if D is empty.
receive(self, datad)	Updates the dict with given datad dictionary.
receive_append(self, resultd)	Appends given resultd dict to list fields in this dict.
setdefault(self, key[, default])	Insert key with a value of default if key is not in the dictionary.
update()	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
values()	

receive (*self*, *datad*)

Updates the dict with given datad dictionary.

receive_append (*self*, *resultd*)

Appends given resultd dict to list fields in this dict.

2.5 Preprocess

class sigfeat.preprocess.mix.**MeanMix** (*source*, ***parameters*)

Averages multi channel source e.g stereo to mono mix.

Parameters

axis [int]

channels [int] Number of channels will be generated by this preprocess.

Attributes

metadata Returns metadata.

parameters Returns all parameters.

Methods

<code>add_metadata(self, name, value)</code>	Appends the key value pair to metadata list.
<code>extend_metadata(self, mdata)</code>	Extends the metadata with the given list of key value pairs.
<code>fetch_metadata_as_attrs(self)</code>	Sets metadata as attributes of self.
<code>generate(self)</code>	Override this method.
<code><i>process</i>(self, data)</code>	Override this method.
<code>unroll_parameters(self, parameters)</code>	This method must be called to collect all parameters and provide them as attributes.

<code>get_class_parameters</code>	
-----------------------------------	--

process (*self*, *data*)

Override this method.

Parameters

data [data from source]

class sigfeat.preprocess.mix.**SumMix** (*source*, ***parameters*)

Sums up multi channel source.

Parameters

axis [int]

channels [int] Number of channels will be generated by this preprocess.

Attributes

metadata Returns metadata.

parameters Returns all parameters.

Methods

<code>add_metadata(self, name, value)</code>	Appends the key value pair to metadata list.
--	--

Continued on next page

Table 22 – continued from previous page

<code>extend_metadata(self, mdata)</code>	Extends the metadata with the given list of key value pairs.
<code>fetch_metadata_as_attrs(self)</code>	Sets metadata as attributes of self.
<code>generate(self)</code>	Override this method.
<code><i>process</i>(self, data)</code>	Override this method.
<code>unroll_parameters(self, parameters)</code>	This method must be called to collect all parameters and provide them as attributes.

<code>get_class_parameters</code>	
-----------------------------------	--

process (*self*, *data*)

Override this method.

Parameters

data [data from source]

Contains the abstract base classes.

3.1 Source

Implements abstract Source base class

```
class sigfeat.base.source.Source (**parameters)
    Base Source Class.
```

The parameters are mandatory for all kinds of sources. Every source must implement a `.generate()` method returning a generator that yields the signal blocks. Source must have `samplerate` and `channels` as metadata.

Parameters

blocksize [int]
overlap [int]
samplerate [scalar]
channels [int]

Attributes

metadata Returns metadata.
parameters Returns all parameters.

Methods

<code>add_metadata(self, name, value)</code>	Appends the key value pair to metadata list.
<code>extend_metadata(self, mdata)</code>	Extends the metadata with the given list of key value pairs.

Continued on next page

Table 1 – continued from previous page

<code>fetch_metadata_as_attrs(self)</code>	Sets metadata as attributes of self.
<code>generate(self)</code>	Override this method.
<code>unroll_parameters(self, parameters)</code>	This method must be called to collect all parameters and provide them as attributes.

<code>get_class_parameters</code>	
-----------------------------------	--

generate (*self*)

Override this method. It must yield data.

Usually data = (block, index)

3.2 Feature

This module implements the abstract base Feature class.

Feature is subclassed for implementing Signal Features.

TODO: What happens if i have a hidden feature but add a instance of tha same that is not hidden? Mus be solved in the FeatureSet! TODO: Handle labels for multidimensional feature output.

class sigfeat.base.feature.**Feature** (*name=None, requirements=None, **parameters*)

Abstract Feature Base Class

Parameters

name [str]

requirements [Feature instances iterable] To override those returned by self.requires().

Notes

At least the *process()* method must be overridden. If your implemented feature depends on the results of another feature, you must override the *requires()* method returning an iterable e.g. list of feature instances.

Attributes

fid Returns the feature identifying tuple.

hidden Returns whether the feature is hidden or not.

metadata Returns metadata.

parameters Returns all parameters.

Methods

<code>add_metadata(self, name, value)</code>	Appends the key value pair to metadata list.
<code>dependencies(self)</code>	Yields all dependencies of this feature.
<code>extend_metadata(self, mdata)</code>	Extends the metadata with the given list of key value pairs.
<code>featureset(self[, new, autoinst, err_missing])</code>	Returns an ordered dict of all features unique in name.

Continued on next page

Table 2 – continued from previous page

<code>fetch_metadata_as_attrs(self)</code>	Sets metadata as attributes of self.
<code>gen_dependencies_instances(self[, autoinst, ...])</code>	Checks deps for being instance or class and yields instances.
<code>hide(self[, b])</code>	Hide the feature.
<code>new(self)</code>	Returns new initial feature instance with same parameters.
<code>on_finished(self, source, featureset, sink)</code>	Override this method to be run after extraction.
<code>on_start(self, source, featureset, sink)</code>	Override this method if your feature needs some initialization.
<code>process(self, data, result)</code>	Override this method returning process results.
<code>requires(self)</code>	Override this method if your feature depends on other features.
<code>unroll_parameters(self, parameters)</code>	This method must be called to collect all parameters and provide them as attributes.
<code>validate_name(self)</code>	Checks for uniqueness of feature name in all dependent features.

<code>get_class_parameters</code>	
-----------------------------------	--

dependencies (*self*)

Yields all dependencies of this feature.

featureset (*self*, *new=False*, *autoinst=False*, *err_missing=True*)

Returns an ordered dict of all features unique in name.

The dict is ordered by the dependency tree order.

Parameters

new [boolean] All features will be reinitialized.

Returns

featdict [OrderedDict] Keys are *fid* and values are feature instances.

fid

Returns the feature identifying tuple.

gen_dependencies_instances (*self*, *autoinst=False*, *err_missing=True*)

Checks deps for being instance or class and yields instances.

hidden

Returns whether the feature is hidden or not.

hide (*self*, *b=True*)

Hide the feature.

new (*self*)

Returns new initial feature instance with same parameters.

on_finished (*self*, *source*, *featureset*, *sink*)

Override this method to be run after extraction.

Parameters

source [source]

featureset [OrderedDict of features]

sink [Sink]

on_start (*self*, *source*, *featureset*, *sink*)

Override this method if your feature needs some initialization.

The Extractor will give you source, featureset and sink.

Parameters

source [source]

featureset [OrderedDict of features]

sink [Sink]

process (*self*, *data*, *result*)

Override this method returning process results.

The data from the source will be in *data* (usually *data* = *block*, *index*, ...).

The required results from other features will be in the *result* argument (dict like *Result()*).

Parameters

data [block, index] Or the data generated from your source.

result [Result] The results from other features of current data. If you provide your required features in the *requires()* method, you will be able to access the results from those features.

Returns

res [e.g. scalar, ndarray or other custom types.] This is the feature result for the current data (block).

requires (*self*)

Override this method if your feature depends on other features.

You can return Feature classes and Feature instances you need for your feature. Yielding is also allowed.

validate_name (*self*)

Checks for uniqueness of feature name in all dependent features.

class sigfeat.base.feature.**HiddenFeature** (*name=None*, *requirements=None*, ***parameters*)

Attributes

fid Returns the feature identifying tuple.

hidden Returns whether the feature is hidden or not.

metadata Returns metadata.

parameters Returns all parameters.

Methods

<code>add_metadata(self, name, value)</code>	Appends the key value pair to metadata list.
<code>dependencies(self)</code>	Yields all dependencies of this feature.
<code>extend_metadata(self, mdata)</code>	Extends the metadata with the given list of key value pairs.
<code>featureset(self[, new, autoinst, err_missing])</code>	Returns an ordered dict of all features unique in name.

Continued on next page

Table 3 – continued from previous page

<code>fetch_metadata_as_attrs(self)</code>	Sets metadata as attributes of self.
<code>gen_dependencies_instances(self[, autoinst, ...])</code>	Checks deps for being instance or class and yields instances.
<code>hide(self[, b])</code>	Hide the feature.
<code>new(self)</code>	Returns new initial feature instance with same parameters.
<code>on_finished(self, source, featureset, sink)</code>	Override this method to be run after extraction.
<code>on_start(self, source, featureset, sink)</code>	Override this method if your feature needs some initialization.
<code>process(self, data, result)</code>	Override this method returning process results.
<code>requires(self)</code>	Override this method if your feature depends on other features.
<code>unroll_parameters(self, parameters)</code>	This method must be called to collect all parameters and provide them as attributes.
<code>validate_name(self)</code>	Checks for uniqueness of feature name in all dependent features.

<code>get_class_parameters</code>	
-----------------------------------	--

`sigfeat.base.feature.features_to_featureset` (*features*, *new=False*, *autoinst=False*)

Returns a featureset of given features distinct in names.

Parameters

features [iterable]

new [reinitialize features as new instances.]

autoinst [auto initialize missing feature classes if required.]

3.3 Result

class `sigfeat.base.result.Result`

Result dict. Behaves 'immutable' to the Feature.process method.

Just a simple dict to hold the results from features.

Methods

<code>clear()</code>	
<code>copy()</code>	
<code>fromkeys(iterable[, value])</code>	Create a new dictionary with keys from iterable and values set to value.
<code>get(self, key[, default])</code>	Return the value for key if key is in the dictionary, else default.
<code>items()</code>	
<code>keys()</code>	
<code>pop()</code>	If key is not found, d is returned if given, otherwise KeyError is raised
<code>popitem()</code>	2-tuple; but raise KeyError if D is empty.

Continued on next page

Table 4 – continued from previous page

<code>setdefault(self, key[, default])</code>	Insert key with a value of default if key is not in the dictionary.
<code>update()</code>	If E is present and has a <code>.keys()</code> method, then does: for k in E: <code>D[k] = E[k]</code> If E is present and lacks a <code>.keys()</code> method, then does: for k, v in E: <code>D[k] = v</code> In either case, this is followed by: for k in F: <code>D[k] = F[k]</code>
<code>values()</code>	

3.4 Sink

Implements the abstract Sink class.

class `sigfeat.base.sink.Sink`
Sink base class.

Methods

<code>receive(datad)</code>	Shall receive dictionaries directly written to source.
<code>receive_append(resultd)</code>	Shall receive result dictionaries appending data to fields.

classmethod `receive(datad)`
Shall receive dictionaries directly written to source.

classmethod `receive_append(resultd)`
Shall receive result dictionaries appending data to fields.

3.5 Preprocess

Implements a simple preprocess base class.

For processing blocks of a source before extracting features. Preprocess behaves like a Source and is consumed by extractor or another Preprocess.

class `sigfeat.base.preprocess.Preprocess(source, **parameters)`
Preprocess base class.
Behaves like a source. But you must override the process method.

Examples

```
>>> src = YourPreprocess(YourSource(...))
>>> extractor.extract(src, ...)
```

Attributes

metadata Returns metadata.

parameters Returns all parameters.

Methods

<code>add_metadata(self, name, value)</code>	Appends the key value pair to metadata list.
<code>extend_metadata(self, mdata)</code>	Extends the metadata with the given list of key value pairs.
<code>fetch_metadata_as_attrs(self)</code>	Sets metadata as attributes of self.
<code><i>generate</i>(self)</code>	Override this method.
<code><i>process</i>(self, data)</code>	Override this method.
<code>unroll_parameters(self, parameters)</code>	This method must be called to collect all parameters and provide them as attributes.

<code>get_class_parameters</code>	
-----------------------------------	--

generate (*self*)

Override this method. It must yield data.

Usually `data = (block, index)`

process (*self*, *data*)

Override this method.

Parameters

data [data from source]

3.6 Parameter

This module implements the Parameter and it's mixin class.

Purpose is, to distinguish parameters of objects from other attributes. Parameters of instances will be extracted into Sink.

class `sigfeat.base.parameter.Parameter` (*default=None*)

Adds a Parameter to the class.

Attributes

default Returns the default value.

Methods

<code><i>validate</i>(self, value)</code>	You can override this method for validation of parameter values.
---	--

default

Returns the default value.

validate (*self*, *value*)

You can override this method for validation of parameter values.

class `sigfeat.base.parameter.ParameterMixin` (***parameters*)

ParameterMixin class

Adds Parameter functionality to classes.

Attributes

parameters Returns all parameters.

Methods

<i>unroll_parameters</i> (self, parameters)	This method must be called to collect all parameters and provide them as attributes.
---	--

<code>get_class_parameters</code>	
-----------------------------------	--

parameters

Returns all parameters.

unroll_parameters (*self, parameters*)

This method must be called to collect all parameters and provide them as attributes.

By calling this function defined parameters will become attributes with values not beeing of type Parameter anymore. But all parameters will be placed in self._parameters as well.

3.7 Metadata

Mixin for Metadata.

Adds methods `add_metadata()`, `extend_metadata()` and `fetch_metadata_as_attrs()` to the class.

Metadata will be extracted into Sink. So created feature datasets contain metadata for e.g. features and sources.

class sigfeat.base.metadata.**MetadataMixin**

MetadataMixin class Adding metadata functionality to classes. Overrides:

- `._metadata`
- `.metadata`
- `._init_metadata_list`
- `.add_metadata`
- `.extend_metadata`
- `.fetch_metadata_as_attrs`

Attributes

metadata Returns metadata.

Methods

<i>add_metadata</i> (self, name, value)	Appends the key value pair to metadata list.
<i>extend_metadata</i> (self, mdata)	Extends the metadata with the given list of key value pairs.
<i>fetch_metadata_as_attrs</i> (self)	Sets metadata as attributes of self.

add_metadata (*self*, *name*, *value*)

Appends the key value pair to metadata list.

extend_metadata (*self*, *mdata*)

Extends the metadata with the given list of key value pairs.

fetch_metadata_as_attrs (*self*)

Sets metadata as attributes of self.

metadata

Returns metadata.

CHAPTER 4

Indices and tables

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

Copyright (c) 2017, Siegfried Gündert All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of SigFeat nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL SIEGFRIED GÜNDERT BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

S

- `sigfeat.base.feature`, 32
- `sigfeat.base.metadata`, 38
- `sigfeat.base.parameter`, 37
- `sigfeat.base.preprocess`, 36
- `sigfeat.base.result`, 35
- `sigfeat.base.sink`, 36
- `sigfeat.base.source`, 31
- `sigfeat.extractor`, 27
- `sigfeat.feature`, 7
- `sigfeat.preprocess.mix`, 29
- `sigfeat.sink.default`, 28
- `sigfeat.source.array`, 5
- `sigfeat.source.soundfile`, 6

A

`add_metadata()` (*sigfeat.base.metadata.MetadataMixin* method), 39

`ArraySource` (class in *sigfeat.source.array*), 5

C

`CrestFactor` (class in *sigfeat.feature*), 22

D

`default` (*sigfeat.base.parameter.Parameter* attribute), 37

`DefaultDictSink` (class in *sigfeat.sink.default*), 28

`Delta` (class in *sigfeat.feature*), 26

`dependencies()` (*sigfeat.base.feature.Feature* method), 33

E

`extend_metadata()` (*sigfeat.base.metadata.MetadataMixin* method), 39

`extract()` (*sigfeat.extractor.Extractor* method), 27

`Extractor` (class in *sigfeat.extractor*), 27

F

`Feature` (class in *sigfeat.base.feature*), 32

`features_to_featureset()` (in module *sigfeat.base.feature*), 35

`featureset()` (*sigfeat.base.feature.Feature* method), 33

`fetch_metadata_as_attrs()` (*sigfeat.base.metadata.MetadataMixin* method), 39

`fid` (*sigfeat.base.feature.Feature* attribute), 33

G

`gen_dependencies_instances()` (*sigfeat.base.feature.Feature* method), 33

`generate()` (*sigfeat.base.preprocess.Preprocess* method), 37

`generate()` (*sigfeat.base.source.Source* method), 32

`generate()` (*sigfeat.source.array.ArraySource* method), 6

`generate()` (*sigfeat.source.soundfile.SoundFileSource* method), 6

`get_features_parameters_and_metadata()` (*sigfeat.extractor.Extractor* method), 27

`get_parameters_and_metadata()` (*sigfeat.extractor.Extractor* static method), 28

H

`hidden` (*sigfeat.base.feature.Feature* attribute), 33

`HiddenFeature` (class in *sigfeat.base.feature*), 34

`hide()` (*sigfeat.base.feature.Feature* method), 33

I

`Index` (class in *sigfeat.feature*), 7

M

`MeanMix` (class in *sigfeat.preprocess.mix*), 29

`metadata` (*sigfeat.base.metadata.MetadataMixin* attribute), 39

`MetadataMixin` (class in *sigfeat.base.metadata*), 38

`MFCC` (class in *sigfeat.feature*), 20

N

`new()` (*sigfeat.base.feature.Feature* method), 33

O

`on_finished()` (*sigfeat.base.feature.Feature* method), 33

`on_start()` (*sigfeat.base.feature.Feature* method), 34

`on_start()` (*sigfeat.feature.SpectralCentroid* method), 11

`on_start()` (*sigfeat.feature.SpectralFlux* method), 9

`on_start()` (*sigfeat.feature.SpectralRolloff* method), 10

`on_start()` (*sigfeat.feature.SpectralSlope* method), 19

`on_start()` (*sigfeat.feature.ZeroCrossingRate method*), 24

P

`Parameter` (*class in sigfeat.base.parameter*), 37

`ParameterMixin` (*class in sigfeat.base.parameter*), 37

`parameters` (*sigfeat.base.parameter.ParameterMixin attribute*), 38

`Peak` (*class in sigfeat.feature*), 24

`Preprocess` (*class in sigfeat.base.preprocess*), 36

`process()` (*sigfeat.base.feature.Feature method*), 34

`process()` (*sigfeat.base.preprocess.Preprocess method*), 37

`process()` (*sigfeat.feature.CrestFactor method*), 23

`process()` (*sigfeat.feature.Delta method*), 26

`process()` (*sigfeat.feature.Index method*), 7

`process()` (*sigfeat.feature.MFCC method*), 20

`process()` (*sigfeat.feature.Peak method*), 25

`process()` (*sigfeat.feature.RootMeanSquare method*), 22

`process()` (*sigfeat.feature.SpectralCentroid method*), 12

`process()` (*sigfeat.feature.SpectralCrestFactor method*), 18

`process()` (*sigfeat.feature.SpectralFlatness method*), 17

`process()` (*sigfeat.feature.SpectralFlux method*), 9

`process()` (*sigfeat.feature.SpectralKurtosis method*), 15

`process()` (*sigfeat.feature.SpectralRolloff method*), 10

`process()` (*sigfeat.feature.SpectralSkewness method*), 14

`process()` (*sigfeat.feature.SpectralSlope method*), 19

`process()` (*sigfeat.feature.SpectralSpread method*), 13

`process()` (*sigfeat.feature.ZeroCrossingRate method*), 24

`process()` (*sigfeat.preprocess.mix.MeanMix method*), 29

`process()` (*sigfeat.preprocess.mix.SumMix method*), 30

R

`receive()` (*sigfeat.base.sink.Sink class method*), 36

`receive()` (*sigfeat.sink.default.DefaultDictSink method*), 28

`receive_append()` (*sigfeat.base.sink.Sink class method*), 36

`receive_append()` (*sigfeat.sink.default.DefaultDictSink method*), 28

`requires()` (*sigfeat.base.feature.Feature method*), 34

`requires()` (*sigfeat.feature.CrestFactor method*), 23

`requires()` (*sigfeat.feature.Delta method*), 27

`requires()` (*sigfeat.feature.MFCC method*), 21

`requires()` (*sigfeat.feature.Peak method*), 26

`requires()` (*sigfeat.feature.RootMeanSquare method*), 22

`requires()` (*sigfeat.feature.SpectralCentroid method*), 12

`requires()` (*sigfeat.feature.SpectralCrestFactor method*), 18

`requires()` (*sigfeat.feature.SpectralFlatness method*), 17

`requires()` (*sigfeat.feature.SpectralFlux method*), 9

`requires()` (*sigfeat.feature.SpectralKurtosis method*), 16

`requires()` (*sigfeat.feature.SpectralRolloff method*), 10

`requires()` (*sigfeat.feature.SpectralSkewness method*), 14

`requires()` (*sigfeat.feature.SpectralSlope method*), 19

`requires()` (*sigfeat.feature.SpectralSpread method*), 13

`reset()` (*sigfeat.extractor.Extractor method*), 28

`Result` (*class in sigfeat.base.result*), 35

`RootMeanSquare` (*class in sigfeat.feature*), 21

S

`sigfeat.base.feature` (*module*), 32

`sigfeat.base.metadata` (*module*), 38

`sigfeat.base.parameter` (*module*), 37

`sigfeat.base.preprocess` (*module*), 36

`sigfeat.base.result` (*module*), 35

`sigfeat.base.sink` (*module*), 36

`sigfeat.base.source` (*module*), 31

`sigfeat.extractor` (*module*), 27

`sigfeat.feature` (*module*), 7

`sigfeat.preprocess.mix` (*module*), 29

`sigfeat.sink.default` (*module*), 28

`sigfeat.source.array` (*module*), 5

`sigfeat.source.soundfile` (*module*), 6

`Sink` (*class in sigfeat.base.sink*), 36

`SoundFileSource` (*class in sigfeat.source.soundfile*), 6

`Source` (*class in sigfeat.base.source*), 31

`SpectralCentroid` (*class in sigfeat.feature*), 11

`SpectralCrestFactor` (*class in sigfeat.feature*), 17

`SpectralFlatness` (*class in sigfeat.feature*), 16

`SpectralFlux` (*class in sigfeat.feature*), 8

`SpectralKurtosis` (*class in sigfeat.feature*), 15

`SpectralRolloff` (*class in sigfeat.feature*), 9

`SpectralSkewness` (*class in sigfeat.feature*), 13

`SpectralSlope` (*class in sigfeat.feature*), 18

`SpectralSpread` (*class in sigfeat.feature*), 12

`SumMix` (*class in sigfeat.preprocess.mix*), 29

U

`unroll_parameters()`

*(sigfeat.base.parameter.ParameterMixin
method)*, [38](#)

V

`validate()` *(sigfeat.base.parameter.Parameter
method)*, [37](#)

`validate_name()` *(sigfeat.base.feature.Feature
method)*, [34](#)

Z

`ZeroCrossingRate` *(class in sigfeat.feature)*, [23](#)