SpiNNUtils Documentation

Release 6.0.0

Apr 07, 2021

Contents

These pages document the python code for the SpiNNUtils module which is part of the SpiNNaker Project (Combined_documentation).

CHAPTER 1

Contents

1.1 spinn_utilities

1.1.1 spinn_utilities package

1.1.1.1 Subpackages

1.1.1.1.1 spinn_utilities.citation package

1.1.1.1.1.1 Module contents

class spinn_utilities.citation.**CitationAggregator**

Bases: object

Helper class for building a citation file which references all dependencies

create_aggregated_citation_file(*module_to_start_at*, *aggregated_citation_file*)

Entrance method for building the aggregated citation file

Parameters

- **module_to_start_at** (*python module*) – the top level module to figure out its citation file for
- **aggregated_citation_file** (*str*) – file name of aggregated citation file

static locate_path_for_c_dependency(*true_software_name*)

Parameters **true_software_name** (*str*) –

Return type str or None

class spinn_utilities.citation.**CitationUpdaterAndDoiGenerator**

Bases: object

static convert_month_name_to_number(*version_month*)
Convert a python month in text form to a number form

Parameters **version_month** (*str or int*) – the text form of the month

Returns the month int value

Return type int

Raises Exception when the month name is not recognised

static convert_text_date_to_date(*version_month*, *version_year*, *version_day*)
Convert the 3 components of a date into a CFF date

Parameters

- **version_month** (*str or int*) – version month, in text form
- **version_year** (*int*) – version year
- **version_day** (*int*) – version day of month

Returns the string representation for the cff file

Return type str

update_citation_file_and_create_doi(*citation_file_path*, *doi_title*, *create_doi*, *publish_doi*, *previous_doi*, *zenodo_access_token*, *module_path*)
Take a CITATION.cff file and updates the version and date-released fields, and rewrites the `CITATION.cff` file.

Parameters

- **citation_file_path** (*str*) – File path to the `CITATION.cff` file
- **create_doi** (*bool*) – Whether to use Zenodo DOI interface to grab a DOI
- **zenodo_access_token** (*str*) – Access token for Zenodo
- **publish_doi** (*bool*) – Whether to publish the DOI on Zenodo
- **previous_doi** (*str*) – DOI to append the created DOI to
- **doi_title** (*str*) – Title for the created DOI
- **module_path** (*str*) – Path to the module to zip up
- **update_version** (*bool*) – Whether we should update the citation version

`spinn_utilities.citation.`**generate_aggregate**(*arguments=None*)
Command-line tool to generate a single `citation.cff` from others.

Parameters **arguments** (*list(str)*) – Command line arguments.

- `--output_path`: Where to write the aggregate file
- `--top_module`: The module to start aggregating the citation.cffs from
- `--doi_title`: The title of the DOI
- `--zenodo_access_token`: The access token for Zenodo
- `--tools_doi`: The DOI of the tools

1.1.1.1.2 spinn_utilities.configs package

1.1.1.1.2.1 Module contents

class spinn_utilities.configs.**CamelCaseConfigParser**(*defaults=None*, *none_marker='None'*)

Bases: configparser.RawConfigParser

get_bool(*section*, *option*)
Get the boolean value of an option.

Parameters

- **section** (*str*) – What section to get the option from.
- **option** (*str*) – What option to read.

Returns The option value.

Return type bool

get_float(*section*, *option*)
Get the float value of an option.

Parameters

- **section** (*str*) – What section to get the option from.
- **option** (*str*) – What option to read.

Returns The option value.

Return type float

get_int(*section*, *option*)
Get the integer value of an option.

Parameters

- **section** (*str*) – What section to get the option from.
- **option** (*str*) – What option to read.

Returns The option value

Return type int

get_str(*section*, *option*)
Get the string value of an option.

Parameters

- **section** (*str*) – What section to get the option from.
- **option** (*str*) – What option to read.

Returns The option value

Return type str or None

get_str_list(*section*, *option*, *token=', '*)
Get the string value of an option split into a list

Parameters

- **section** (*str*) – What section to get the option from.

- **option** (*str*) – What option to read.
- **token** – The token to split the string into a list

Returns The list (possibly empty) of the option values

Return type list(str)

optionxform(*optionstr*)
Transforms the name of an option to lower case and strips underscores, so matching is more user-friendly.

read(*filenames*, *encoding=None*)
Read and parse a filename or a list of filenames.

read_files
The configuration files that have been actually read.

class spinn_utilities.configs.**CaseSensitiveParser**(*defaults=None*, *dict_type=<class 'collections.OrderedDict'>*, *allow_no_value=False*, ***, *delimiters=('=', ':')*, *comment_prefixes=('#', ';')*, *inline_comment_prefixes=None*, *strict=True*, *empty_lines_in_values=True*, *default_section='DEFAULT'*, *interpolation=<object object>*, *converters=<object object>*)
Bases: configparser.RawConfigParser

optionxform(*optionstr*)
Performs no transformation of option strings.

exception spinn_utilities.configs.**NoConfigFoundException**
Bases: Exception

Throws when an existing Section has an extra config value

exception spinn_utilities.configs.**UnexpectedConfigException**
Bases: Exception

Throws when an existing Section has an extra config value

1.1.1.1.3 spinn_utilities.make_tools package

1.1.1.1.3.1 Submodules

1.1.1.1.3.2 spinn_utilities.make_tools.replacer module

class spinn_utilities.make_tools.replacer.**Replacer**(*dict_pointer*)
Bases: object

Performs replacements.

Parameters **dict_pointer** (*str*) – Where to find the dictionary file

replace(*short*)
Apply the replacements to a short message.

Parameters **short** (*str*) – The short message to apply the transform to.

Returns The expanded message.

Return type str

1.1.1.1.3.3 Module contents

class spinn_utilities.make_tools.**Converter**(*src*, *dest*, *dict_file*)
Bases: object

Converts a whole directory including sub directories

Parameters

- **src** (*str*) – Full source directory
- **dest** (*str*) – Full destination directory
- **dict_file** (*str*) – Full path to dictionary file

static convert(*src*, *dest*, *dict_file*)
Wrapper function around this class.

run()
Runs the file converter on a whole directory including sub-directories.

> **Warning:** This code is absolutely not thread safe. Interwoven calls even on different FileConverter objects is dangerous! It is highly likely that dict files become corrupted and the same `message_id` is used multiple times.

class spinn_utilities.make_tools.**FileConverter**(*src*, *dest*, *dict_file*)
Bases: object

Creates the file_convertor to convert one file

Parameters

- **src** (*str*) – Source file
- **dest** (*str*) – Destination file
- **dict_file** (*str*) – File to hold dictionary mappings

bracket_count(*text*)
Net count of open brackets in line.

Parameters **text** (*str*) –

Return type int

static convert(*src*, *dest*, *dict_file*, *range_start=1*)
Static method to create Object and do the conversion

Parameters

- **src** (*str*) – Source file
- **dest** (*str*) – Destination file
- **dict_file** (*str*) – File to hold dictionary mappings
- **range_start** (*int*) – id of last dictionary key used

Returns The last message id use which can in turn be passed into this method again (`range_start`) to get contiguous non-overlapping IDs across many files.

Return type int

dest
Full destination file name

Type str

dict
File to hold dictionary mappings

Type str

quote_part(*text*)
Net count of double quotes in line.

Parameters **text** (*str*) –

Return type int

split_by_comma_plus(*main*, *line_num*)
split line by comma and partially parse

Parameters

- **main** (*str*) –
- **line_num** (*int*) –

Return type list(str)

src
Full source file name

Type str

unique_src()
Returns the suffix of the source and destination paths which is the same.

For example, assuming sources of `/spinnaker/sPyNNaker/neural_modelling/src/common/in_spikes.h` `/spinnaker/sPyNNaker/neural_modelling/modified_src/common/in_spikes.h` this returns `src/common/in_spikes.h`

Returns A pointer to the source relative to the destination

Return type str

class spinn_utilities.make_tools.**Replacer**(*dict_pointer*)
Bases: object

Performs replacements.

Parameters **dict_pointer** (*str*) – Where to find the dictionary file

replace(*short*)
Apply the replacements to a short message.

Parameters **short** (*str*) – The short message to apply the transform to.

Returns The expanded message.

Return type str

1.1.1.1.4 spinn_utilities.matrix package

1.1.1.1.4.1 Module contents

class spinn_utilities.matrix.**AbstractMatrix**
Bases: object

A rectangular 2D collection of data.

get_data(*x*, *y*)
Get the value at a particular X,Y coordinate.

set_data(*x*, *y*, *value*)
Set the value at a particular X,Y coordinate.

class spinn_utilities.matrix.**DemoMatrix**
Bases: spinn_utilities.matrix.abstract_matrix.AbstractMatrix

data

get_data(*x*, *y*)
Get the value at a particular X,Y coordinate.

set_data(*x*, *y*, *value*)
Set the value at a particular X,Y coordinate.

class spinn_utilities.matrix.**DoubleDict**(*xtype*, *ytype*, *matrix*)
Bases: object

class spinn_utilities.matrix.**XView**(*x*, *matrix*)
Bases: object

A view along a particular x-slice of a 2D matrix.

class spinn_utilities.matrix.**YView**(*y*, *matrix*)
Bases: object

A view along a particular y-slice of a 2D matrix.

1.1.1.1.5 spinn_utilities.ranged package

1.1.1.1.5.1 Module contents

An implementation of a dictionary and a list that support efficiently workingwith ranges of values, used to implement efficient collections for PyNNpopulation views and assemblies.

class spinn_utilities.ranged.**AbstractDict**
Bases: object

Base class for the *RangeDictionary* and *all* views. This allows the users to not have to worry if they have a view.

get_default(*key*)
Gets the default value for a single key. Unless changed, the default is the original value.

Note: Does not change any values but only changes what reset_value would do

Parameters **key** (*str*) – Existing dict key

Returns default for this key.

get_ranges(*key=None*)
Lists the ranges(s) for all IDs covered by this view. There will be one yield for each range which may cover one or more IDs.

Note: As the data is created in a single call this is not affected by any updates.

Parameters **key** (*str or iterable(str) or None*) – The key or keys to get the value of. Use None for all

Returns List of tuples of (*start*, *stop*, *value*). *start* is *inclusive* so is the first ID in the range. *stop* is *exclusive* so is the last ID in the range + 1. If *key* is a str, *value* is a single object. If *key* is iterable (list, tuple, set, etc) of str (or None) *value* is a dictionary object

get_value(*key*)
Gets a single shared value for all IDs covered by this view.

Parameters **key** (*str or iterable(str) or None*) – The key or keys to get the value of. Use None for all

Returns If key is a str, this returns the single object. If key is iterable (list, tuple, set, etc) of str (or None), returns a dictionary object

Raises ***MultipleValuesException*** – If even one of the keys has multiple values set. But not if other keys not asked for have multiple values

has_key(*key*)
As the Deprecated dict `has_keys` function.

Note: Int keys to IDs are not supported.

Parameters **key** (*str*) – the key

Returns If the key is in dict

Return type bool

ids()
Returns the IDs in range or view. If the view is setup with IDs out of numerical order the order used to create the view is maintained.

Note: If indexing into a view, you are picking the X'th ID. So if the IDs are *[2,3,4,5]* the *view[2]* will be the data for ID *4* and not *2*

Returns list of IDs

Return type list(int)

items()
Returns a list of (`key`, `value`) tuples. Works only if the whole ranges/view has single values.

If the key is a str, the values are single objects. If the key is iterable (list, tuple, set, etc) of str (or None), the values are dictionary objects

Returns List of (`key`, `value`) tuples

Raises ***MultipleValuesException*** – If even one of the keys has multiple values set.

iter_all_values(*key*, *update_save=False*)

Iterates over the value(s) for all IDs covered by this view. There will be one yield for each ID even if values are repeated.

Parameters

- **key** (*str or iterable(str) or None*) – The key or keys to get the value of. Use None for all keys
- **update_save** – If set True the iteration will work even if values are updated during iteration. If left False the iterator may be faster but behaviour is *undefined* and *unchecked* if *any* values are changed during iteration.

Returns If key is a str, this yields single objects. If key is iterable (list, tuple, set, etc) of str (or None), yields dictionary objects

iter_ranges(*key=None*)

Iterates over the ranges(s) for all IDs covered by this view. There will be one yield for each range which may cover one or more IDs.

> **Warning:** This iterator is *not* update safe! Behaviour is *undefined* and *unchecked* if *any* values are changed during iteration.

Parameters **key** (*str or iterable(str) or None*) – The key or keys to get the value of. Use None for all

Returns yields tuples of (*start*, *stop*, *value*). *start* is *inclusive* so is the first ID in the range. *stop* is *exclusive* so is the last ID in the range + 1. If *key* is a str, *value* is a single object. If *key* is iterable (list, tuple, set, etc) of str (or None), *value* is a dictionary object

iteritems()

Iterates over the (`key`, `value`) tuples. Works only if the whole ranges/view has single values.

If the key is a str, the values are single objects. If the key is iterable (list, tuple, set, etc) of str (or None), the values are dictionary objects

This function is safe for value updates but may miss new keys added during iteration.

Returns yield (`key`, `value`) tuples

Raises ***MultipleValuesException*** – If even one of the keys has multiple values set.

itervalues()

Iterates over the values. Works only if the whole ranges/view has single values.

If key is a str, the values are single objects. If key is iterable (list, tuple, set, etc) of str (or None), values are dictionary objects

This function is safe for value updates but may miss new keys added during iteration.

Returns yield values

Raises ***MultipleValuesException*** – If even one of the keys has multiple values set.

keys()

Returns the keys in the dictionary

Returns keys in the dict

reset(*key*)
Sets the value(s) for a single key back to the default value.

Parameters

- **key** (*str*) – Existing dict key
- **default** – Value to be used by reset

set_value(*key*, *value*, *use_list_as_value=False*)
Resets a already existing key to the new value. All IDs in the whole range or view will have this key set.

> **Warning:** This method does not allow adding keys. Using *dict[str] =* will add a new key, but it is not supported for views.

> **Warning:** If a View is created over multiple ranges this method would raise a *KeyError* if any the ranges does not have the key. (Currently multiple ranges not yet supported.)

Parameters

- **key** (*str*) – key to value being set
- **value** – any object
- **use_list_as_value** – True if the value *is* a list

Raises **KeyError** – If a new key is being used.

values()
Returns a list of values. Works only if the whole ranges/view has single values.

If key is a str, the values are single objects. If key is iterable (list, tuple, set, etc) of str (or None), values are dictionary objects

Returns List of values

Raises ***MultipleValuesException*** – If even one of the keys has multiple values set.

class spinn_utilities.ranged.**AbstractList**(*size*, *key=None*)
Bases: spinn_utilities.ranged.abstract_sized.AbstractSized

A ranged implementation of list.

Functions that change the size of the list are *not* supported. These include:

```
__setitem__ where key >= len
__delitem__
append
extend
insert
pop
remove
```

Function that manipulate the list based on values are not supported. These include:

```
reverse, __reversed__
sort
```

In the current version the IDs are zero base consecutive numbers so there is no difference between value-based IDs and index-based IDs but this could change in the future.

Supports the following arithmetic operations over the list:

+ element-wise addition or addition of a single scalar

- element-wise subtraction or subtraction of a single scalar

***** element-wise multiplication or multiplication by a single scalar

/ element-wise true division or true division by a single scalar

// element-wise floor division or floor division by a single scalar

Parameters

- **size** – Fixed length of the list
- **key** – The dict key this list covers. This is used only for better Exception messages

apply_operation(*operation*)

Applies a function on the list to create a new one. The values of the new list are created on the fly so any changes to the original lists are reflected.

Parameters **operation** – A function that can be applied over the individual values to create new ones.

Returns new list

Return type *AbstractList*

count(*x*)

Counts the number of elements in the list with value `x`

Parameters **x** –

Returns

get_default()

Gets the default value of the list. Just in case we later allow to increase the number of elements.

Returns Default value

get_single_value_all()

If possible, returns a single value shared by the whole list.

For multiple values use `for x in list`, `iter(list)` or `list.iter`, or one of the `iter_ranges` methods

Returns Value shared by all elements in the list

Raises ***MultipleValuesException*** – If even one elements has a different value

get_single_value_by_ids(*ids*)

If possible, returns a single value shared by all the IDs.

For multiple values, use `for x in list`, `iter(list)`, `list.iter`, or one of the `iter_ranges` methods.

Returns Value shared by all elements with these IDs

Raises ***MultipleValuesException*** – If even one elements has a different value. Not thrown if elements outside of the IDs have a different value, even if these elements are between the ones pointed to by IDs

get_single_value_by_slice(*slice_start*, *slice_stop*)
If possible, returns a single value shared by the whole slice list.

For multiple values, use `for x in list`, `iter(list)`, `list.iter`, or one of the `iter_ranges` methods

Returns Value shared by all elements in the slice

Raises ***MultipleValuesException*** – If even one elements has a different value. Not thrown if elements outside of the slice have a different value

get_value_by_id(*id*)
Returns the value for one item in the list

Parameters **id** (*int*) – One of the IDs of an element in the list

Returns The value of that element

get_values(*selector=None*)
Get the value all elements pointed to the selector.

Note: Unlike `__get_item__` this method always returns a list even if the selector is a single int

Parameters **selector** – See *AbstractSized.selector_to_ids()*

Returns returns a list if the item which may be empty or have only single value

Return type list

index(*x*)
Finds the first ID of the first element in the list with value `x`

Parameters **x** –

Returns

iter()
Update-safe iterator of all elements.

Note: Duplicate/Repeated elements are yielded for each ID

Returns yields each element one by one

iter_by_id(*id*)
Fast but *not* update-safe iterator by one ID.

While `next` can only be called once, this is an iterator so it can be mixed in with other iterators.

Parameters **id** – ID

Returns yields the elements

iter_by_ids(*ids*)
Fast but *not* update-safe iterator by collection of IDs.

Note: Duplicate/Repeated elements are yielded for each ID.

Parameters **ids** – IDs

Returns yields the elements pointed to by IDs

iter_by_selector(*selector=None*)
Fast but *not* update-safe iterator of all elements in the slice.

Parameters **selector** – See *AbstractSized.selector_to_ids()*

Returns yields the selected elements one by one

iter_by_slice(*slice_start*, *slice_stop*)
Fast but *not* update-safe iterator of all elements in the slice.

Note: Duplicate/Repeated elements are yielded for each ID

Returns yields each element one by one

iter_ranges()
Fast but *not* update-safe iterator of the ranges.

Returns yields each range one by one

iter_ranges_by_id(*id*)
Iterator of the range for this ID

Note: The start and stop of the range will be reduced to just the ID

This method purpose is so one a control method can select which iterator to use.

Returns yields the one range

iter_ranges_by_ids(*ids*)
Fast but *not* update-safe iterator of the ranges covered by these IDs.

For consecutive IDs where the elements have the same value a single range may be yielded.

Note: The start and stop of the range will be reduced to just the IDs

Returns yields each range one by one

iter_ranges_by_slice(*slice_start*, *slice_stop*)
Fast but *not* update-safe iterator of the ranges covered by this slice.

Note: The start and stop of the range will be reduced to just the IDs inside the slice.

Returns yields each range one by one

range_based()
Shows if the list is suited to deal with ranges or not.

All list must implement all the range functions, but there are times when using ranges will probably be slower than using individual values. For example the individual values may be stored in a list in which case the ranges are created on demand.

Returns True if and only if Ranged based calls are recommended.

Return type bool

class spinn_utilities.ranged.**DualList**(*left*, *right*, *operation*, *key=None*)
Bases: spinn_utilities.ranged.abstract_list.AbstractList

A list which combines two other lists with an operation.

Parameters

- **left** (AbstractList) – The first list to combine
- **right** (AbstractList) – The second list to combine
- **operation** (*callable*) – The operation to perform as a function that takes two values and returns the result of the operation
- **key** – The dict key this list covers. This is used only for better Exception messages

get_default()
Gets the default value of the list. Just in case we later allow to increase the number of elements.

Returns Default value

get_single_value_by_ids(*ids*)
If possible, returns a single value shared by all the IDs.

For multiple values, use `for x in list`, `iter(list)`, `list.iter`, or one of the `iter_ranges` methods.

Returns Value shared by all elements with these IDs

Raises ***MultipleValuesException*** – If even one elements has a different value. Not thrown if elements outside of the IDs have a different value, even if these elements are between the ones pointed to by IDs

get_single_value_by_slice(*slice_start*, *slice_stop*)
If possible, returns a single value shared by the whole slice list.

For multiple values, use `for x in list`, `iter(list)`, `list.iter`, or one of the `iter_ranges` methods

Returns Value shared by all elements in the slice

Raises ***MultipleValuesException*** – If even one elements has a different value. Not thrown if elements outside of the slice have a different value

get_value_by_id(*id*)
Returns the value for one item in the list

Parameters **id** (*int*) – One of the IDs of an element in the list

Returns The value of that element

iter_by_slice(*slice_start*, *slice_stop*)
Fast but *not* update-safe iterator of all elements in the slice.

Note: Duplicate/Repeated elements are yielded for each ID

Returns yields each element one by one

iter_ranges()
Fast but *not* update-safe iterator of the ranges.

Returns yields each range one by one

iter_ranges_by_slice(*slice_start*, *slice_stop*)
Fast but *not* update-safe iterator of the ranges covered by this slice.

Note: The start and stop of the range will be reduced to just the IDs inside the slice.

Returns yields each range one by one

range_based()
Shows if the list is suited to deal with ranges or not.

All list must implement all the range functions, but there are times when using ranges will probably be slower than using individual values. For example the individual values may be stored in a list in which case the ranges are created on demand.

Returns True if and only if Ranged based calls are recommended.

Return type bool

class spinn_utilities.ranged.**SingleList**(*a_list*, *operation*, *key=None*)
Bases: spinn_utilities.ranged.abstract_list.AbstractList

A List that performs an operation on the elements of another list.

Parameters

- **a_list** (AbstractList) – The list to perform the operation on
- **operation** (*callable*) – A function which takes a single value and returns the result of the operation on that value
- **key** – The dict key this list covers. This is used only for better Exception messages

get_default()
Gets the default value of the list. Just in case we later allow to increase the number of elements.

Returns Default value

get_single_value_by_ids(*ids*)
If possible, returns a single value shared by all the IDs.

For multiple values, use `for x in list`, `iter(list)`, `list.iter`, or one of the `iter_ranges` methods.

Returns Value shared by all elements with these IDs

Raises ***MultipleValuesException*** – If even one elements has a different value. Not thrown if elements outside of the IDs have a different value, even if these elements are between the ones pointed to by IDs

get_single_value_by_slice(*slice_start*, *slice_stop*)
If possible, returns a single value shared by the whole slice list.

For multiple values, use `for x in list`, `iter(list)`, `list.iter`, or one of the `iter_ranges` methods

Returns Value shared by all elements in the slice

Raises ***MultipleValuesException*** – If even one elements has a different value. Not thrown if elements outside of the slice have a different value

get_value_by_id(*id*)
Returns the value for one item in the list

Parameters **id** (*int*) – One of the IDs of an element in the list

Returns The value of that element

iter_ranges()
Fast but *not* update-safe iterator of the ranges.

Returns yields each range one by one

iter_ranges_by_slice(*slice_start*, *slice_stop*)
Fast but *not* update-safe iterator of the ranges covered by this slice.

Note: The start and stop of the range will be reduced to just the IDs inside the slice.

Returns yields each range one by one

range_based()
Shows if the list is suited to deal with ranges or not.

All list must implement all the range functions, but there are times when using ranges will probably be slower than using individual values. For example the individual values may be stored in a list in which case the ranges are created on demand.

Returns True if and only if Ranged based calls are recommended.

Return type bool

class spinn_utilities.ranged.**AbstractSized**(*size*)
Bases: object

Base class for slice and ID checking against size.

Parameters **size** – Fixed length of the list

selector_to_ids(*selector*, *warn=False*)
Gets the list of IDs covered by this selector. The types of selector currently supported are:

None: Returns all IDs.

slice: Standard python slice. Negative values and values larger than size are handled using slices's indices method. This could result in am empty list.

int: (or long) Handles negative values as normal. Checks if ID is within expected range.

iterator of bools: Used as a mask. If the length of the mask is longer or shorted than number of IDs the result is the shorter of the two, with the remainder of the longer ignored.

iterator of int (long) but not bool: Every value checked that it is with the range 0 to size. Negative values are *not* allowed. Original order and duplication is respected so result may be unordered and contain duplicates.

Parameters

- **selector** – Some object that identifies a range of IDs.
- **warn** – If True, this method will warn about problems with the selector.

Returns a (possibly sorted) list of IDs

class spinn_utilities.ranged.**AbstractView**(*range_dict*)
Bases: spinn_utilities.ranged.abstract_dict.AbstractDict

A view over a ranged dictionary.

Note: The view may currently be read from only with int and int-collection indices, and only be written to with str indices. This may change to become more permissive in future versions.

Use *RangeDictionary.view_factory()* to create views

get_default(*key*)
Gets the default value for a single key. Unless changed, the default is the original value.

Note: Does not change any values but only changes what `reset_value` would do

Parameters **key** (*str*) – Existing dict key

Returns default for this key.

keys()
Returns the keys in the dictionary

Returns keys in the dict

exception spinn_utilities.ranged.**MultipleValuesException**(*key=None*, *value1=None*, *value2=None*)
Bases: Exception

class spinn_utilities.ranged.**RangeDictionary**(*size*, *defaults=None*)
Bases: spinn_utilities.ranged.abstract_sized.AbstractSized, spinn_utilities.ranged.abstract_dict.AbstractDict

Main holding class for a range of similar Dictionary object. Keys in the dictionary must be str object and can not be removed. New keys can be added using the `dict[str] = value` format. The size (length of the list) is fixed and set at initialisation time.

The Object is set up initially where every ID in the range will share the same value for each key. All keys must be of type str. The default Values can be anything including None.

Parameters

- **size** (*int*) – Fixed number of IDs / Length of lists
- **defaults** (*dict*) – Default dictionary where all keys must be str

copy()

copy_into(*other*)
Turns this dict into a copy of the other dict but keep its id

Parameters **other** (*RangedDict*) – Another Ranged Dictionary assumed created by cloning this one

get_default(*key*)
Gets the default value for a single key. Unless changed, the default is the original value.

Note: Does not change any values but only changes what `reset_value` would do

Parameters **key** (*str*) – Existing dict key

Returns default for this key.

get_list(*key*)
Gets the storage unit for a single key.

Note: Mainly intended by Views to access the data for one key directly.

Parameters **key** (*str*) – a key which must be present in the dict

Return type ranged_list.RangedList

get_value(*key=None*)
Gets a single shared value for all IDs covered by this view.

Parameters **key** (*str or iterable(str) or None*) – The key or keys to get the value of. Use None for all

Returns If key is a str, this returns the single object. If key is iterable (list, tuple, set, etc) of str (or None), returns a dictionary object

Raises ***MultipleValuesException*** – If even one of the keys has multiple values set. But not if other keys not asked for have multiple values

get_values_by_id(*key*, *id*)
Same as *get_value()* but limited to a single ID.

Parameters

- **key** – as *get_value()*
- **id** – single int ID

Returns See *get_value()*

has_key(*key*)
As the Deprecated dict has_keys function.

Note: Int keys to IDs are not supported.

Parameters **key** (*str*) – the key

Returns If the key is in dict

Return type bool

ids()
Returns the IDs in range or view. If the view is setup with IDs out of numerical order the order used to create the view is maintained.

Note: If indexing into a view, you are picking the X'th ID. So if the IDs are *[2,3,4,5]* the *view[2]* will be the data for ID *4* and not *2*

Returns list of IDs

Return type list(int) Returns a list of the IDs in this Range

Returns a list of the IDs in this Range

Return type list(int)

iter_all_values(*key=None*, *update_save=False*)
Iterates over the value(s) for all IDs covered by this view. There will be one yield for each ID even if values are repeated.

Parameters

- **key** (*str or iterable(str) or None*) – The key or keys to get the value of. Use None for all keys
- **update_save** – If set True the iteration will work even if values are updated during iteration. If left False the iterator may be faster but behaviour is *undefined* and *unchecked* if *any* values are changed during iteration.

Returns If key is a str, this yields single objects. If key is iterable (list, tuple, set, etc) of str (or None), yields dictionary objects

iter_ranges(*key=None*)
Iterates over the ranges(s) for all IDs covered by this view. There will be one yield for each range which may cover one or more IDs.

> **Warning:** This iterator is *not* update safe! Behaviour is *undefined* and *unchecked* if *any* values are changed during iteration.

Parameters **key** (*str or iterable(str) or None*) – The key or keys to get the value of. Use None for all

Returns yields tuples of (*start*, *stop*, *value*). *start* is *inclusive* so is the first ID in the range. *stop* is *exclusive* so is the last ID in the range + 1. If *key* is a str, *value* is a single object. If *key* is iterable (list, tuple, set, etc) of str (or None), *value* is a dictionary object

iter_ranges_by_id(*key=None*, *id=None*)
Same as *iter_ranges()* but limited to one ID.

Parameters

- **key** – see *iter_ranges()* parameter key
- **id** (*int*) – single ID which is the actual ID and not an index into IDs

iter_ranges_by_ids(*ids*, *key=None*)
Same as *iter_ranges()* but limited to a collection of IDs.

The IDs are actual ID values and not indexes into the IDs

Parameters

- **key** – see *iter_ranges()* parameter `key`
- **ids** – Collection of IDs in the range

Returns see *iter_ranges()*

iter_ranges_by_slice(*key*, *slice_start*, *slice_stop*)
Same as *iter_ranges()* but limited to a simple slice.

`slice_start` and `slice_stop` are actual ID values and not indexes into the IDs. They must also be actual values, so `None`, `max_int`, and negative numbers are not supported.

Parameters

- **key** – see *iter_ranges()* parameter `key`

- **slice_start** – Inclusive i.e. first ID
- **slice_stop** – Exclusive to last ID + 1

Returns see *iter_ranges()*

iter_values_by_ids(*ids*, *key=None*, *update_save=False*)
Same as *iter_all_values()* but limited to a simple slice.

iter_values_by_slice(*slice_start*, *slice_stop*, *key=None*, *update_save=False*)
Same as *iter_all_values()* but limited to a simple slice.

keys()
Returns the keys in the dictionary

Returns keys in the dict

list_factory(*size*, *value*, *key*)
Defines which class or subclass of *RangedList* to use.

Main purpose is for subclasses to use a subclass or RangedList. All parameters are pass through ones to the List constructor

Parameters

- **size** – Fixed length of the list
- **value** – value to given to all elements in the list
- **key** – The dict key this list covers.

Returns AbstractList in this case a RangedList

set_default(*key*, *default*)
Sets the default value for a single key.

Note: Does not change any values but only changes what `reset_value` would do

Warning: If called on a View it sets the default for the *whole* range and not just the view.

Parameters

- **key** (*str*) – Existing dict key
- **default** – Value to be used by reset

set_value(*key*, *value*, *use_list_as_value=False*)
Resets a already existing key to the new value. All IDs in the whole range or view will have this key set.

Warning: This method does not allow adding keys. Using *dict[str] =* will add a new key, but it is not supported for views.

Warning: If a View is created over multiple ranges this method would raise a *KeyError* if any the ranges does not have the key. (Currently multiple ranges not yet supported.)

Parameters

- **key** (*str*) – key to value being set
- **value** – any object
- **use_list_as_value** – True if the value *is* a list

Raises **KeyError** – If a new key is being used.

update_safe_iter_all_values(*key*, *ids*)
Same as *iter_all_values()* but limited to a collection of IDs and update-safe.

view_factory(*key*)
Main function for creating views. This is the preferred way of creating new views as it checks parameters and returns the most efficient view.

Note the __getitem__ methods called by Object[id] and similar defer to this method so are fine to use.

The ID(s) used are the actual IDs in the range and not indexes on the list of IDs

Parameters **key** – A single int ID, a Slice object, or an iterable of int IDs

Returns A view over the range

class spinn_utilities.ranged.**RangedList**(*size=None*, *value=None*, *key=None*, *use_list_as_value=False*)
Bases: spinn_utilities.ranged.abstract_list.AbstractList

A list that is able to efficiently hold large numbers of elements that all have the same value.

Parameters

- **size** (*int or None*) – Fixed length of the list; if None, the value must be a sized object.
- **value** (*object or Sized*) – value to given to all elements in the list
- **key** – The dict key this list covers. This is used only for better Exception messages
- **use_list_as_value** (*bool*) – True if the value *is* a list

static as_list(*value*, *size*, *ids=None*)
Converts (if required) the value into a list of a given size. An exception is raised if value cannot be given size elements.

Note: This method can be extended to add other conversions to list in which case *is_list()* must also be extended.

Parameters **value** –

Returns value as a list

Raises **Exception** – if the number of values and the size do not match

copy()
Creates a copy of this list.

Depth is just enough so that any changes done through the RangedList API on other will not change self

Returns The copy

Return type *RangedList*

copy_into(*other*)
Turns this List into a of the other list but keep its id

Depth is just enough so that any changes done through the RangedList API on other will not change self

Parameters **other** (RangedList) – Another Ranged List to copy the values from

get_default()
Returns the default value for this list.

Returns Default Value

Return type object

get_ranges()
Returns a copy of the list of ranges.

Note: As this is a copy it will not reflect any updates.

Return type list(tuple(int,int,object))

get_single_value_by_ids(*ids*)
If possible, returns a single value shared by all the IDs.

For multiple values, use `for x in list`, `iter(list)`, `list.iter`, or one of the `iter_ranges` methods.

Returns Value shared by all elements with these IDs

Raises ***MultipleValuesException*** – If even one elements has a different value. Not thrown if elements outside of the IDs have a different value, even if these elements are between the ones pointed to by IDs

get_single_value_by_slice(*slice_start*, *slice_stop*)
If possible, returns a single value shared by the whole slice list.

For multiple values, use `for x in list`, `iter(list)`, `list.iter`, or one of the `iter_ranges` methods

Returns Value shared by all elements in the slice

Raises ***MultipleValuesException*** – If even one elements has a different value. Not thrown if elements outside of the slice have a different value

get_value_by_id(*id*)
Returns the value for one item in the list

Parameters **id** (*int*) – One of the IDs of an element in the list

Returns The value of that element

static is_list(*value*, *size*)
Determines if the value should be treated as a list.

Note: This method can be extended to add other checks for list in which case *as_list()* must also be extended.

iter_by_slice(*slice_start*, *slice_stop*)
Fast but *not* update-safe iterator of all elements in the slice.

Note: Duplicate/Repeated elements are yielded for each ID

Returns yields each element one by one

iter_ranges()
Fast but *not* update-safe iterator of the ranges.

Returns yields each range one by one

iter_ranges_by_slice(*slice_start*, *slice_stop*)
Fast but *not* update-safe iterator of the ranges covered by this slice.

Note: The start and stop of the range will be reduced to just the IDs inside the slice.

Returns yields each range one by one

range_based()
Shows if the list is suited to deal with ranges or not.

All list must implement all the range functions, but there are times when using ranges will probably be slower than using individual values. For example the individual values may be stored in a list in which case the ranges are created on demand.

Returns True if and only if Ranged based calls are recommended.

Return type bool

set_default(*default*)
Sets the default value.

Note: Does not change the value of any element in the list.

Parameters **default** (*object*) – new default value

set_value(*value*, *use_list_as_value=False*)
Sets *all* elements in the list to this value.

Note: Does not change the default.

Parameters

- **value** – new value
- **use_list_as_value** – True if the value to be set *is* a list

set_value_by_id(*id*, *value*)
Sets the value for a single ID to the new value.

Note: This method only works for a single positive int ID. Use `set` or `__set__` for slices, tuples, lists and negative indexes.

Parameters

- **id** (*int*) – Single ID
- **value** (*object*) – The value to save

set_value_by_ids(*ids*, *value*, *use_list_as_value=False*)

set_value_by_selector(*selector*, *value*, *use_list_as_value=False*)

Support for the `list[x] =` format.

Parameters

- **selector** (*int or slice or list(int)*) – A single ID, a slice of IDs or a list of IDs
- **value** (*object*) –

set_value_by_slice(*slice_start*, *slice_stop*, *value*, *use_list_as_value=False*)

Sets the value for a single range to the new value.

Note: This method only works for a single positive range. Use `set` or `__set__` for slices, tuples, lists and negative indexes.

Parameters

- **slice_start** (*int*) – Start of the range
- **slice_stop** (*int*) – Exclusive end of the range
- **value** (*object*) – The value to save

class spinn_utilities.ranged.**RangedListOfList**(*size=None*, *value=None*, *key=None*, *use_list_as_value=False*)

Bases: `spinn_utilities.ranged.ranged_list.RangedList`

Parameters

- **size** (*int or None*) – Fixed length of the list; if `None`, the value must be a sized object.
- **value** (*object or Sized*) – value to given to all elements in the list
- **key** – The dict key this list covers. This is used only for better Exception messages
- **use_list_as_value** (*bool*) – True if the value *is* a list

static is_list(*value*, *size*)

Determines if the value should be treated as a list.

Note: This method can be extended to add other checks for list in which case `as_list()` must also be extended.

1.1.1.1.6 spinn_utilities.testing package

1.1.1.1.6.1 Submodules

1.1.1.1.6.2 spinn_utilities.testing.log_checker module

spinn_utilities.testing.log_checker.**assert_logs_contains_once**(*level*, *log_records*, *message*)

Checks if the log records contain exactly one record at the given level with the given sub-message.

Note: While this code does not depend on testfixtures, you will need testfixtures to generate the input data

Parameters

- **level** – The log level. Probably "INFO", "WARNING" or "ERROR".
- **log_records** – list of log records returned by testfixtures.LogCapture
- **submessage** – String which should be part of a log record

Return type None

Raises **AssertionError** – If the submessage is not present in the log

spinn_utilities.testing.log_checker.**assert_logs_error_contains**(*log_records*, *submessage*)

Checks it the log records contain an ERROR log with this sub-message

Note: While this code does not depend on testfixtures, you will need testfixtures to generate the input data

Parameters

- **log_records** – list of log records returned by testfixtures.LogCapture
- **submessage** – String which should be part of an ERROR log

Return type None

Raises **AssertionError** – If the submessage is not present in the log

spinn_utilities.testing.log_checker.**assert_logs_error_not_contains**(*log_records*, *submessage*)

Checks it the log records do not contain an ERROR log with this sub-message

Note: While this code does not depend on testfixtures, you will need testfixtures to generate the input data

Parameters

- **log_records** – list of log records returned by testfixtures.LogCapture
- **submessage** – String which should be part of an ERROR log

Return type None

Raises **AssertionError** – If the submessage is present in the log

spinn_utilities.testing.log_checker.**assert_logs_info_contains**(*log_records*, *sub_message*)

Checks it the log records contain an INFO log with this sub-message

Note: While this code does not depend on testfixtures, you will need testfixtures to generate the input data

Parameters

- **log_records** – list of log records returned by testfixtures.LogCapture
- **sub_message** – String which should be part of an INFO log

Return type None

Raises **AssertionError** – If the submessage is not present in the log

spinn_utilities.testing.log_checker.**assert_logs_info_not_contains**(*log_records*, *submessage*)

Checks it the log records do not contain an INFO log with this sub-message

Note: While this code does not depend on testfixtures, you will need testfixtures to generate the input data

Parameters

- **log_records** – list of log records returned by testfixtures.LogCapture
- **submessage** – String which should be part of an INFO log

Return type None

Raises **AssertionError** – If the submessage is present in the log

1.1.1.1.6.3 Module contents

1.1.1.2 Submodules

1.1.1.3 spinn_utilities.abstract_base module

A trimmed down version of standard Python Abstract Base classes.

If using `@add_metaclass`, this requires:

```
from six import add_metaclass
```

Using Python 3 style `metaclass=AbstractBase` is preferred.

class spinn_utilities.abstract_base.**AbstractBase**

Bases: type

Metaclass for defining Abstract Base Classes (AbstractBases).

Use this metaclass to create an AbstractBase. An AbstractBase can be subclassed directly, and then acts as a mix-in class.

This is a trimmed down version of ABC. Unlike ABC you can not register unrelated concrete classes.

spinn_utilities.abstract_base.**abstractmethod**(*funcobj*)

A decorator indicating abstract methods.

Requires that the metaclass is *AbstractBase* or derived from it. A class that has a metaclass derived from *AbstractBase* cannot be instantiated unless all of its abstract methods are overridden. The abstract methods can be called using any of the normal 'super' call mechanisms.

Usage:

```
@add_metaclass(AbstractBase)
class C:
    @abstractmethod
    def my_abstract_method(self, ...):
        ...

# Python 3 only syntax
class C3(object, metaclass=AbstractBase):
    @abstractmethod
    def my_abstract_method(self, ...):
        ...
```

class spinn_utilities.abstract_base.**abstractproperty**

Bases: property

A decorator indicating abstract properties.

Requires that the metaclass is *AbstractBase* or derived from it. A class that has a metaclass derived from *AbstractBase* cannot be instantiated unless all of its abstract properties are overridden. The abstract properties can be called using any of the normal 'super' call mechanisms.

Usage:

```
@add_metaclass(AbstractBase)
class C:
    @abstractproperty
    def my_abstract_property(self):
        ...

# Python 3 only syntax
class C3(object, metaclass=AbstractBase):
    @abstractproperty
    def my_abstract_property(self):
        ...
```

This defines a read-only property; you can also define a read-write abstract property using the 'long' form of property declaration:

```
@add_metaclass(AbstractBase)
class C:
    def getx(self): ...
    def setx(self, value): ...
    x = abstractproperty(getx, setx)

# Python 3 only syntax
class C3(object, metaclass=AbstractBase):
    def getx(self): ...
    def setx(self, value): ...
    x = abstractproperty(getx, setx)
```

1.1.1.4 spinn_utilities.abstract_context_manager module

class spinn_utilities.abstract_context_manager.**AbstractContextManager**
Bases: object

Closeable class that supports being used as a simple context manager.

close()
How to actually close the underlying resources.

1.1.1.5 spinn_utilities.bytestring_utils module

spinn_utilities.bytestring_utils.**as_hex**(*bytestring*, *start=None*, *end=None*)
Returns the bytestring as string showing the hex values

Parameters

- **bytestring** – data as a byteString
- **start** – the inclusive start of the slice to return. May be None
- **end** – the exclusive end of the slice to return. May be None

Returns Comma separated hex values

spinn_utilities.bytestring_utils.**as_string**(*bytestring*, *start=None*, *end=None*)
Returns the length and the hex values.

The length is always the full length irrespective of the start and end.

Parameters

- **bytestring** – data as a bytestring
- **start** – the inclusive start of the slice to return. May be None
- **end** – the exclusive end of the slice to return. May be None

Returns The length of the bytesting and the hex values, comma separated

1.1.1.6 spinn_utilities.classproperty module

class spinn_utilities.classproperty.**ClassPropertyDescriptor**(*fget*)
Bases: object

A class to handle the management of class properties

spinn_utilities.classproperty.**classproperty**(*func*)
Defines a property at the class-level

1.1.1.7 spinn_utilities.conf_loader module

spinn_utilities.conf_loader.**install_cfg_and_IOError**(*filename*, *defaults*, *config_locations*)
Installs a local configuration file based on the templates and raises an exception.

This method is called when no user configuration file is found.

It will create a file in the users home directory based on the defaults. Then it prints a helpful message and throws an error with the same message.

Parameters

- **filename** (*str*) – Name under which to save the new configuration file
- **defaults** (*list(str)*) – List of full paths to the default configuration files. Each of which *must* have an associated template file with exactly the same path plus *.template*.
- **config_locations** (*list(str)*) – List of paths where the user configuration files were looked for. Only used for the message

Raises ***spinn_utilities.configs.NoConfigFoundException*** – Always raised

spinn_utilities.conf_loader.**load_config**(*filename*, *defaults*, *config_parsers=None*, *validation_cfg=None*)

Load the configuration.

Parameters

- **filename** (*str*) – The base name of the configuration file(s). Should not include any path components.
- **defaults** (*list(str)*) – The list of files to get default configurations from.
- **config_parsers** (*list(tuple(str, ConfigParser))*) – The parsers to parse the sections of the configuration file with, as a list of (section name, parser); a configuration section will only be parsed if the section_name is found in the configuration files already loaded. The standard logging parser is appended to (a copy of) this list.
- **validation_cfg** (*list(str)*) – The list of files to read a validation configuration from. If omitted, no such validation is performed.

Returns the fully-loaded and checked configuration

spinn_utilities.conf_loader.**logging_parser**(*config*)

Create the root logger with the given level.

Create filters based on logging levels

Note: You do not normally need to call this function; it is used automatically to parse Logging configuration sections.

1.1.1.8 spinn_utilities.default_ordered_dict module

class spinn_utilities.default_ordered_dict.**DefaultOrderedDict**(*default_factory=None*, **args*, ***kwargs*)

Bases: collections.OrderedDict

copy() → a shallow copy of od

1.1.1.9 spinn_utilities.exceptions module

exception spinn_utilities.exceptions.**FailedToFindBinaryException**

Bases: *spinn_utilities.exceptions.SpiNNUtilsException*

Raised when the executable finder cant find the binary

exception spinn_utilities.exceptions.**SpiNNUtilsException**

Bases: Exception

Superclass of all exceptions from the SpiNNUtils module.

1.1.1.10 spinn_utilities.executable_finder module

class spinn_utilities.executable_finder.**ExecutableFinder**(*binary_search_paths*)
Bases: object

Manages a set of folders in which to search for binaries, and allows for binaries to be discovered within this path

Parameters **binary_search_paths** (*iterable(str)*) – The initial set of folders to search for binaries.

add_path(*path*)
Adds a path to the set of folders to be searched. The path is added to the end of the list, so it is searched after all the paths currently in the list.

Parameters **path** (*str*) – The path to add

binary_paths
The set of folders to search for binaries, as a printable colon-separated string.

Return type str

check_logs()

clear_logs()

get_executable_path(*executable_name*)
Finds an executable within the set of folders. The set of folders is searched sequentially and the first match is returned.

Parameters **executable_name** (*str*) – The name of the executable to find

Returns The full path of the discovered executable

Return type str

Raises **KeyError** – If no executable was found in the set of folders

get_executable_paths(*executable_names*)
Finds each executables within the set of folders.

The names are assumed to be comma separated The set of folders is searched sequentially and the first match for each name is returned.

Names not found are ignored and not added to the list.

Parameters **executable_names** (*str*) – The name of the executable to find. Assumed to be comma separated.

Returns The full path of the discovered executable, or `None` if no executable was found in the set of folders

Return type list(str)

1.1.1.11 spinn_utilities.find_max_success module

spinn_utilities.find_max_success.**find_max_success**(*max_possible*, *check*)
Finds the maximum value that will pass the check

Parameters

- **max_possible** (*int*) – The maximum value that should be tested.
- **check** – A boolean function that given an int value returns true for every value up and including the cutoff and false for every value greater than the cutoff

Returns The highest value that returns true for the check but is not more than the max_possible

spinn_utilities.find_max_success.**search_for_max_success**(*best_success*, *min_fail*, *check*)

Finds the maximun value in the range that pass the check

Parameters

- **best_success** (*int*) – A minimum value that needs not be tested because it is either known to succeed or is a flag for total failure. Can be negative
- **min_fail** (*int*) – A maximum value that needs not be tested because it is either known to fail or one more than the maximum interesting value but must be greater than best_success but may also be negative
- **check** – A boolean function that given an int value returns true for every value up and including the cutoff and false for every value greater than the cutoff

Returns The highest value that returns true in the range between best_success and min_fail (both exclusive ends) or best_success if the whole range fails or is empty.

1.1.1.12 spinn_utilities.helpful_functions module

spinn_utilities.helpful_functions.**gcd**(**numbers*)

Greatest common divisor of 1 or more integers.

GIGO: If any of the values are anything except positive int values this function will either produce incorrect results or raise an exception.

Parameters **numbers** – The Positive integers to get the GCD for. This can be one or more int values or a singleton which is an iterator (not empty) of ints.

Returns the gcd or 1 if numbers is empty or an empty iterator

Return type int

Raises

- **TypeError** – If any value can not be interpreted as an Integer or if no values are provided
- **ZeroDivisionError** – May be raised if one of the values is zero

spinn_utilities.helpful_functions.**get_valid_components**(*module*, *terminator*)

Get possible components, stripping the given suffix from their class names.

Parameters

- **module** – The module containing the classes to obtain.
- **terminator** (*str*) – Regular expression string to match the suffix. Anchoring not required.

Returns mapping from (shortened) name to class

Return type dict(str -> class)

spinn_utilities.helpful_functions.**is_singleton**(*value*)

Tests whether the value is a singleton.

Singleton types are strings and any other class that can not be iterated.

Strings are considered singleton as rarely will someone use a String to represent an iterable of characters

spinn_utilities.helpful_functions.**lcm**(**numbers*)
Lowest common multiple of 0, 1 or more integers.

GIGO: If any of the values are anything except positive int values this function will either produce incorrect results or raise an exception.

Parameters **numbers** – The Positive integers to get the lcm for. This can be zero, one or more int values or a singleton which is an iterator (possibly empty) of ints.

Returns the lcm or 1 if numbers is empty or an empty iterator

Return type int

Raises

- **TypeError** – If any value can not be interpreted as an Integer
- **ZeroDivisionError** – May be raised if one of the values is zero

1.1.1.13 spinn_utilities.index_is_value module

class spinn_utilities.index_is_value.**IndexIsValue**
Bases: object

Tiny support class that implements `object[x]` by just returning `x` itself.

Used for where you want a range from 1 to *N* but you don't know *N*.

Clearly, operations that assume a finite list are *not* supported.

1.1.1.14 spinn_utilities.log module

class spinn_utilities.log.**ConfiguredFilter**(*conf*)
Bases: object

Allow a parent logger to filter a child logger.

filter(*record*)
Get the level for the deepest parent, and filter appropriately.

class spinn_utilities.log.**ConfiguredFormatter**(*conf*)
Bases: logging.Formatter

Defines the logging format for the SpiNNaker host software.

static construct_logging_parents(*conf*)
Create a dictionary of module names and logging levels.

static deepest_parent(*parents*, *child*)
Greediest match between child and parent.

static level_of_deepest_parent(*parents*, *child*)
The logging level of the greediest match between child and parent.

class spinn_utilities.log.**FormatAdapter**(*logger*, *extra=None*)
Bases: logging.LoggerAdapter

An adaptor for logging with messages that uses Python format strings.

Example:

```
log = FormatAdapter(logging.getLogger(__name__))
log.info("this message has {} inside {}", 123, 'itself')
# --> INFO: this message has 123 inside itself
```

log(*level*, *msg*, **args*, ***kwargs*)
Delegate a log call to the underlying logger, applying appropriate transformations to allow the log message to be written using Python format string, rather than via %-substitutions.

process(*msg*, *kwargs*)
Process the logging message and keyword arguments passed in to a logging call to insert contextual information. You can either manipulate the message itself, the keyword arguments or both. Return the message and *kwargs* modified (or not) to suit your needs.

classmethod set_kill_level(*level=None*)
Allow system to change the level at which a log is changed to an Exception

Static so effects all log messages

Parameters **level** (*int*) – The level to set. The values in `logging` are recommended.

classmethod set_report_File(*report_file*)

Parameters

- **report_file** –
- **write_normal** –

Returns

exception spinn_utilities.log.**LogLevelTooHighException**
Bases: `Exception`

An Exception throw when the System tries to log at a level where an Exception is a better option.

1.1.1.15 spinn_utilities.logger_utils module

spinn_utilities.logger_utils.**error_once**(*logger*, *msg*)

spinn_utilities.logger_utils.**reset**()

spinn_utilities.logger_utils.**warn_once**(*logger*, *msg*)

1.1.1.16 spinn_utilities.ordered_set module

class spinn_utilities.ordered_set.**OrderedSet**(*iterable=None*)
Bases: `collections.abc.MutableSet`

add(*value*)
Add an element.

discard(*value*)
Remove an element. Do not raise an exception if absent.

peek(*last=True*)

pop(*last=True*)
Return the popped value. Raise KeyError if empty.

update(*iterable*)

1.1.1.17 spinn_utilities.overrides module

class spinn_utilities.overrides.**overrides**(*super_class_method*, *extend_doc=True*, *additional_arguments=None*, *extend_defaults=False*)

Bases: object

A decorator for indicating that a method overrides another method in a superclass. This checks that the method does actually exist, copies the doc-string for the method, and enforces that the method overridden is specified, making maintenance easier.

Parameters

- **super_class_method** – The method to override in the superclass
- **extend_doc** (*bool*) – True the method doc string should be appended to the super-method doc string, False if the documentation should be set to the super-method doc string only if there isn't a doc string already
- **additional_arguments** (*iterable(str)*) – Additional arguments taken by the subclass method over the superclass method, e.g., that are to be injected
- **extend_defaults** (*bool*) – Whether the subclass may specify extra defaults for the parameters

1.1.1.18 spinn_utilities.package_loader module

spinn_utilities.package_loader.**all_modules**(*directory*, *prefix*, *remove_pyc_files=False*)

List all the python files found in this directory giving then the prefix.

Any file that ends in either `.py` or `.pyc` is assume a python module and added to the result set.

Parameters

- **directory** (*str*) – path to check for python files
- **prefix** (*str*) – package prefix top add to the file name

Returns set of python package names

Return type set(str)

spinn_utilities.package_loader.**load_module**(*name*, *remove_pyc_files=False*, *exclusions=None*, *gather_errors=True*)

Loads this modules and all its children.

Parameters

- **name** (*str*) – name of the modules
- **remove_pyc_files** (*bool*) – True if `.pyc` files should be deleted
- **exclusions** (*list(str)*) – a list of modules to exclude
- **gather_errors** (*bool*) – True if errors should be gathered, False to report on first error

Returns None

spinn_utilities.package_loader.**load_modules**(*directory*, *prefix*, *remove_pyc_files=False*, *exclusions=None*, *gather_errors=True*)

Loads all the python files found in this directory, giving them the specified prefix

Any file that ends in either `.py` or `.pyc` is assume a python module and added to the result set.

Parameters

- **directory** (*str*) – path to check for python files
- **prefix** (*str*) – package prefix top add to the file name
- **remove_pyc_files** (*bool*) – True if `.pyc` files should be deleted
- **exclusions** (*list(str)*) – a list of modules to exclude
- **gather_errors** (*bool*) – True if errors should be gathered, False to report on first error

Returns None

1.1.1.19 spinn_utilities.ping module

class spinn_utilities.ping.**Ping**

Bases: object

static host_is_reachable(*ipaddr*)

static ping(*ipaddr*)

unreachable = {}

1.1.1.20 spinn_utilities.progress_bar module

class spinn_utilities.progress_bar.**DummyProgressBar**(*total_number_of_things_to_do*, *string_describing_what_being_progressed*, *step_character='='*, *end_character='|'*)

Bases: *spinn_utilities.progress_bar.ProgressBar*

This is a dummy version of the progress bar that just stubs out the internal printing operations with code that does nothing. It otherwise fails in exactly the same way.

class spinn_utilities.progress_bar.**ProgressBar**(*total_number_of_things_to_do*, *string_describing_what_being_progressed*, *step_character='='*, *end_character='|'*)

Bases: object

Progress bar for telling the user where a task is up to

MAX_LENGTH_IN_CHARS = 60

TOO_MANY_ERROR = 'Too many update steps in progress bar! This may be a sign that someth

end()

Close the progress bar, updating whatever is left if needed

Return type None

over(*collection*, *finish_at_end=True*)

Simple wrapper for the cases where the progress bar is being used to show progress through the iteration over a single collection. The progress bar should have been initialised to the size of the collection being iterated over.

Parameters

- **collection** – The base collection (any iterable) being iterated over
- **finish_at_end** (*bool*) – Flag to say if the bar should finish at the end of the collection

Returns An iterable. Expected to be directly used in a for.

update(*amount_to_add=1*)
Update the progress bar by a given amount

Parameters **amount_to_add** –

Return type None

1.1.1.21 spinn_utilities.require_subclass module

spinn_utilities.require_subclass.**require_subclass**(*required_class*)
Decorator that arranges for subclasses of the decorated class to require that they are also subclasses of the given class.

Parameters **required_class** (*type*) – The class that the subclass of the decorated class must be an instance of (if that subclass is concrete).

1.1.1.22 spinn_utilities.safe_eval module

class spinn_utilities.safe_eval.**SafeEval**(**args*, ***kwargs*)
Bases: object

This provides expression evaluation capabilities while allowing the set of symbols exposed to the expression to be strictly controlled.

Sample of use:

```
>>> import math
>>> def evil_func(x):
...     print("HAHA!")
...     return x/0.0
...
>>> eval_safely = SafeEval(math)
>>> eval_safely.eval("math.sqrt(x)", x=1.23)
1.1090536506409416
>>> eval_safely.eval("evil_func(1.23)")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File ".../safe_eval.py", line 62, in eval
    return eval(expression, self._environment, kwargs)
  File "<string>", line 1, in <module>
NameError: name 'evil_func' is not defined
```

Warning: This is not guaranteed to be safe under all circumstances. It is not designed to be a fully secured interface; it just *discourages* abuse.

Parameters

- **args** – The symbols to use to populate the global reference table. Note that all of these symbols must support the *__name__* property, but that includes any function, method of an object, or module. If you want to make an object available by anything other than its inherent name, define it in the *eval()* call.
- **kwargs** – Define the symbols with explicit names. Needed because some symbols (e.g., constants in numpy) do not have names that we can otherwise look up easily.

eval(*expression*, ***kwargs*)
Evaluate an expression and return the result.

Parameters

- **expression** (*str*) – The expression to evaluate
- **kwargs** – The extra symbol bindings to use for this evaluation. This is useful for passing in particular parameters to an individual evaluation run.

Returns The expression result, the type of which will depend on the expression itself and the operations exposed to it.

1.1.1.23 spinn_utilities.see module

class spinn_utilities.see.**see**(*documentation_method*, *extend_doc=True*, *additional_arguments=None*, *extend_defaults=False*)
Bases: *spinn_utilities.overrides.overrides*

A decorator for indicating that the documentation of the method is provided by another method with exactly the same arguments.

Note: This has the same effect as overrides in reality, but is provided to show that the method doesn't actually override

---

1.1.1.24 spinn_utilities.socket_address module

class spinn_utilities.socket_address.**SocketAddress**(*notify_host_name*, *notify_port_no*, *listen_port*)
Bases: object

Data holder for a socket interface for notification protocol.

listen_port
The port to listen to for responses

notify_host_name
The notify host name

notify_port_no
The notify port no

1.1.1.25 spinn_utilities.timer module

class spinn_utilities.timer.**Timer**
Bases: object

A timer used for performance measurements.

Recommended usage:

```
with Timer() as timer:
    ... do stuff that takes time ...

elapsed_time = timer.measured_interval
```

or alternatively:

```
timer = Timer()
timer.start_timing()
... do stuff that takes time ...
elapsed_time = timer.take_sample()
```

Mixing these two styles is *not recommended*.

measured_interval
Get how long elapsed during the measured section.

Return type datetime.timedelta

start_timing()
Start the timing. Use *take_sample()* to get the end.

take_sample()
Describes how long has elapsed since the instance that the *start_timing()* method was last called.

Return type datetime.timedelta

1.1.1.26 Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y